FACULTY OF ENGINEERING AND ARCHITECTURE

Enabling Interactive Querying for Latency-Sensitive Applications on Big Datasets

Leandro Ordonez Ante

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Computer Science Engineering

Supervisors
Prof. Filip De Turck, PhD - Tim Wauters, PhD

Department of Information Technology
Faculty of Engineering and Architecture, Ghent University

March 2022

GHENT UNIVERSITY

ISBN 978-94-6355-576-0
NUR 988, 995
Wettelijk depot: D/2022/10.500/17

Members of the Examination Board

Chair

Prof. Em. Luc Taerwe, PhD, Ghent University

Other members entitled to vote

Prof. Antoon Bronselaer, PhD, Ghent University
Prof. Pieter Colpaert, PhD, Ghent University
Prof. Alexandru Costan, PhD, Institut National des Sciences Appliquées de Rennes, France
Pieter Thysebaert, PhD, TomTom, Germany
Gregory Van Seghbroeck, PhD, Ghent University

Supervisors

Prof. Filip De Turck, PhD, Ghent University
Tim Wauters, PhD, Ghent University

Acknowledgements

"Often I feel I go to some distant region of the world to be reminded of who I really am"

—Michael Crichton, *Travels*

As I am approaching the end of my Ph.D. and reflecting on all the things I got to live these last few years, I cannot help but feel humbled and grateful for the generosity and support of the people that made this all possible. Therefore, it is only fitting for me to start this dissertation by expressing my deepest appreciation to those that helped me get to this point. First, I would like to thank my supervisors. Prof. Filip De Turck and dr. Tim Wauters, thank you for your tremendous support and encouragement, and your advice and guidance throughout the course of my Ph.D. I only hope I was able to reward the trust you have placed in me by granting me this opportunity. Special thanks to dr. Gregory Van Seghbroeck. I appreciate the enlightening discussions we had, especially during the first stages of my research. You would always have the right insights to get me out of the dark and steer me in the right direction. A big thanks to prof. Bruno Volckaert, for providing me with critical feedback by reviewing my papers, and for your expert direction of the projects I was involved in under your supervision. I feel honored to have been able to work with you all.

I am also grateful to the members of the examination board. Prof. Luc Taerwe, prof. Antoon Bronselaer, prof. Pieter Colpaert, prof. Alexandru Costan, dr. Pieter Thysebaert, and dr. Gregory Van Seghbroeck: your feedback and suggestions definitely contributed to further enhancing the quality of this dissertation.

I certainly have to mention prof. Ruben Verborgh. You were one of the first people to believe I had what it takes to do research within IDLab. It was thanks to you that in the cold spring of 2013 I ended up in Ghent doing a research internship at the former Multimedia Lab. During the over two months period I worked under your supervision I got to know a lot of talented people: Anastasia, Dörthe, Joachim, Pieter, Miel, Laurens, Sam, Tom, Baptist, Sebastiaan, Pieterjan, prof. Erik Mannens. The experience I had during those few weeks at the Zuiderpoort was enough to spark my interest in pursuing the Ph.D. that I am finishing. I will be always grateful to you for the opportunity you gave me then.

An exceptional research journey would not be possible without exceptional colleagues. To the fine gentlemen I had the chance to share office 200.212 with: Stefano, Thomas V., Niels, Maxim, Jeroen S., Merlijn, Thijs, Wim, Sander, Jerico, Laurens V.H., Tom, Vincent, Cedric M., my heartfelt appreciation goes to you as well. Thank you for creating such a pleasant environment to work in, for your patience to help me find my way in Belgium, and for lending me a hand whenever I needed it. One of the things I miss the most from the pre-pandemic times is to come to the office and meet you all at lunchtime. You would always have an interesting story to share. Oftentimes, Merlijn would come up with some bizarre and fascinating topic of conversation, leading to endless discussions, sometimes leaving us questioning the very nature of reality. Those were the good old days! My fellow office mates, you will be sorely missed. I am thankful also to my colleagues from the 10th floor, past and present, Rafael, Thomas D., Wannes, José, Ankita, Lucas, Dries, Jasper, Sarah, Laurens D., Mathias, Gilles, Philip, Johannes, Andrés, Alejandro, Priscila, Jeroen v.d.H., Elias, Pieter, Cedric D.B., Sam, Lander, Tim, Joachim and to all the others I got the chance to share a word or two at the numerous events we used to attend together. You made this journey all the more enjoyable. I feel genuinely fortunate to count myself as part of such a remarkable team.

My sincere appreciation goes as well to the kind and hardworking people from the administrative staff at IDLab. Martine, Davinia, Karen, Joke, Bernadette, Mike, thank you for the invaluable support you provide. From the get-go, you made my work at the Lab a lot easier. Also, my gratitude goes to Sabrina and her colleagues from ISS for their help in keeping a clean and comfortable workspace at the iGent building. A big thank you to the leaders of the group headed by prof. Piet Demeester for their relentless work in realizing a thriving research culture and creating a stimulating and supportive environment within IDLab. Also, my research would not have been possible without the expert support from our colleagues from the A-Team. Thank you, Brecht, Joeri, Sai, and Vicent, for your work in streamlining the infrastructure needed for us to run our experiments.

I owe an immense debt of gratitude to the incredible people I have met in Ghent during these years. People I have the fortune to call my friends. Stefano, Rafa, Khaled, Tim, Mabel, Tiago, Cata, Mario, Silvia, Diogo, thank you for each conversation we had, the countless meals and good beers we shared, the BBQs we enjoyed together, sometimes regardless of how challenging the weather outside was. All those amazing moments I treasure dearly. Thank you for the many times you welcomed me into your homes and for helping me settle into my new life in Belgium. Anastasia and Julián, I cannot thank you enough for the incredible support you have meant to me and Liz. Just knowing you guys are around have made the challenging times a lot easier, and the good times a lot more fun. I will always cherish the memories we have made together and look forward to the many more times we will meet again. Julián, hermano, estos años acá no hubieran

sido igual de agradables sin vos. Gracias por tu ayuda, tus consejos, por escucharme cuando lo necesitaba. Es una fortuna para mi que hayamos seguido un camino similar desde hace tantos años.
To the remarkable group of fellow Colombians I happen to meet in Ghent: Alexandra, Daniel, Andrés, Lina, Mónica, Alejandro, Nathaly, Sebastián, Luis, Germán, Carolina, gracias infinitas amigos por las celebraciones, las noches de juegos, la buena comida Colombiana que compartimos frecuentemente, las conversaciones, las risas, las ocasionales sacadas de apuros también, en fin, todos los buenos momentos. Ustedes han sido una parte importante de esta historia.
A mis amigos en Colombia, Luis, Javier, Esteban, Samara, Juan Andrés, Nathaly, y a quienes no alcanzo a nombrar, también a ustedes les debo las gracias por su apoyo y por mantener el contacto en la distancia. Agradecimientos también a mis profesores de la Universidad del Cauca, en especial al prof. Juan Carlos Corrales, al prof. Álvaro Rendón y al prof. Gustavo Ramirez, por brindarme la formación en investigación que me permitió llegar a aplicar a un doctorado en el exterior.
Of course, reaching this point in my Ph.D. trajectory would not have been possible without the abiding love and support of my family: A mi papá, a mi mamá, y a mis hermanitas, todo mi amor y la más profunda gratitud por acompañarme en esta carrera. Gracias por creer en mí y darme la confianza para afrontar los retos que vienen con la vida. Estar lejos de ustedes ha sido una de las cosas más difíciles de esta experiencia, pero cada mensaje y llamada suyos me llenaron de ánimo y fortaleza. Anhelo verlos pronto, celebrar con ustedes, y también compartir—como antes—las cosas simples de la cotidianidad. Agradezco a Don Jaime y Doña Rosalba por el cariño y constante apoyo durante estos años. Para toda mi numerosa familia, quienes han estado alentándome continuamente, a todos ustedes también, Mil Gracias.
Last but not least, to my life partner and wife Liza María, gracias por animarte a tomar mi mano y saltar a lo incierto junto conmigo. Tu valor, tu constancia y tenacidad son una inspiración para mi. Gracias por rescatarme de los días tristes. Gracias por tus cuidados y tu paciencia para soportar mis malos días. Gracias por aventurarte a caminar la vida a mi lado. Te amo inmensamente, este triunfo también es para ti.

Gent, March 2022
Leandro

Table of Contents

List of Figures

List of Tables

List of Acronyms

A

ACID	Atomicity, Consistency, Isolation, Durability
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
AVG	Average

B

BOLD	Big Open Linked Data
Bpost	Belgian Postal service

C

CDN	Content Delivery Network
CoT	City of Things
CPU	Central Process Unit
CQELS	Continuous Query Evaluation over Linked Stream
CRF	Constant Rate Factor
CSV	Comma Separated Values
CTF	Center Tile First

D

DB	Database
DCoB	Dynamic Collective Buffer
DNN	Deep Neural Networks

DRL	Deep Reinforcement Learning
DTW	Dynamic Time Warping

E

EA	Evolutionary Algorithms
ECDF	Empirical Cumulative Distribution Function
ETSI	European Telecommunications Standards Institute
EXPLORA	Efficient eXPLORation through Aggregation

F

FIFO	First In, First Out
FoV	Field of View
FPS	Frames Per Second
FWO	Research Foundation Flanders

G

GB	Gigabytes
Gbps	Gigabits per Second

H

HAS	HTTP Adaptive Streaming
HDFS	Hadoop Distributed File System
HEVC	High Efficiency Video Coding
HMD	HeadMounted Device
HQ	High Quality
HS	HistoricalSpatial Queries
HTTP	Hypertext Transfer Protocol

I

IoT	Internet of Things

IT	Information Technologies

J

JSON	JavaScript Object Notation

L

LD	Linked Data
LDF	Linked Data Fragments
LoRaWAN	Long-Range Wide-Area Network
LRU	LeastRecently Used

M

MB	Megabytes
Mbps	Megabits per Second
MEC	Multiaccess Edge Computing
ML	Machine Learning
MPP	Massively Parallel Processing
MTP	MotiontoPhoton
MVPP	Multiple View Processing Plan

N

NCC	New Cordorcet Criterium
NoSQL	Not Only SQL

O

OLAP	Online Analytical Processing
OLTP	Online Transaction Processing

P

PM	Particle Matter

PSO	Particle Swarm Optimization

Q

QoE	Quality of Experience
QoS	Quality of Service

R

RAM	Random Access Memory
RDBMS	Relational Database Management System
RDF	Resource Description Framework
REST	Representational State Transfer
RSP	RDF Stream Processing
RTT	Round Trip Time

S

SA	Simulated Annealing
SCDAP	Smart City Data Analytics Panel
SF	Scaling Factor
SOSA	Sensor, Observation, Sample, and Actuator
SPARQL	SPARQL Protocol and RDF Query Language
SPJ	Select, Project, Join
SQL	Structured Query Language
SSB	Star Schema Benchmark
SSN	Semantic Sensor Network Ontology
ST	SnapshotTemporal queries
STB	SpaceTime Box
STSF	Summary Time Series Fragments

T

TC	Traffic Control
TCAM	Ternary ContentAddressable Memory
TSAD	TimeSliced Anomaly Detection

U

URI	Uniform Resource Identifier

V

VR	Virtual Reality

W

WET	Workload Execution Time
WKT	WellKnown Text
WPGMA	Weighted Pair Group Method with Arithmetic Mean
WSN	Wireless Sensor Networks
WWW	World Wide Web

Samenvatting
– Summary in Dutch –

Het steeds meer overvloedig aanwezig zijn van dataopslag, rekenkracht en netwerkcapaciteit in de afgelopen decennia heeft geleid tot een explosieve groei van het beschikbare volume aan data, een trend die naar verwachting de komende jaren zal versnellen. Volgens een recente voorspelling van de International Data Corporation (IDC) zal de hoeveelheid data die wereldwijd in 2021 wordt gegenereerd, oplopen tot 79 zettabytes (79×10^{21} bytes). Dit cijfer zal de komende vijf jaar toenemen, waardoor het volume aan data dat wereldwijd is gecreëerd tegen 2025 op 181 zettabytes zal komen te liggen. In de moderne hyperverbonden wereld laat vrijwel elke interactie die we hebben met onze omgeving een digitaal voetspoor achter. Organisaties in alle sectoren wenden zich steeds meer tot technologieën die zich in het domein van het zogenaamde Internet of Things (IoT) bevinden om hun activiteiten te monitoren en data over hun bedrijfsactiviteiten te verzamelen. Naarmate de hoeveelheid, de verscheidenheid en de complexiteit van data groter worden, neemt ook de moeilijkheid toe om er inzichten uit te verwerken, te analyseren en te distilleren. Dergelijke *big data* zijn de mogelijkheden van traditionele technologieën voor databeheer al lang ontgroeid, en hebben de behoefte aan alternatieve mechanismen voor het bewaren van en toegang krijgen tot data op een betrouwbare, schaalbare en performante manier naar voren gebracht. Onderzoek en innovatie op het gebied van het beheer van big data hebben geleid tot de ontwikkeling van een grote verscheidenheid aan oplossingen, elk voor bepaalde specifieke gebruikssituaties. Deze technologieën voor big data variëren van NoSQL-databases die data kunnen bevatten die op verscheidene manieren zijn gestructureerd (bijv. in de vorm van documenten, key-value data, grafen en kolommen), tot gedistribueerde systemen die door gebruik van een commodity hardware cluster data-intensieve workloads op kunnen slaan en uit kunnen voeren.

Voor moderne organisaties is het van het allergrootste belang om toegang te krijgen tot data en er inzichten uit te halen zodra deze zijn gegenereerd om tot de juiste besluitvorming te komen. De noodzaak om de tijd nodig om inzichten uit data te verwerven (*data-to-insight*) te minimaliseren, heeft geleid tot een toenemende interesse in *event-based architectures* en *near real-time stream processing frameworks*. Ondanks hun groeiende

populariteit zijn deze oplossingen nog niet wijdverspreid aanwezig in de bedrijfswereld. De meeste dataverwerking die tegenwoordig wordt uitgevoerd, is nog steeds voornamelijk gebaseerd op methoden geschikt voor de verwerking van batches waarvan bekend is dat ze een aanzienlijke responstijd met zich meebrengen. Hierdoor lopen organisaties het risico beslissingen te nemen en actie te ondernemen op basis van verouderde data, met name voor tijdkritische toepassingen die op deze grote, meerdimensionale dataverzamelingen draaien. Deze thesis heeft daarom als doel om onderzoek te doen naar de realisatie van interactieve query's met een lage responstijd die op grote multidimensionale datasets uitgevoerd worden. Met betrekking tot dit probleem worden in deze thesis vier grote uitdagingen onderzocht:

C1: Dimensionale modellering is een basistechniek voor het structureren van historische bedrijfsdata, waarbij gebruik wordt gemaakt van gedenormaliseerde schema's om snelle verwerking van analytische query's mogelijk te maken. Eenmaal gedefinieerd, bieden deze schema's echter weinig flexibiliteit waardoor het lastig is om ze aan te passen of te optimaliseren wanneer respectievelijk de workload verandert of wanneer tijdrovende query's uitgevoerd worden. Deze thesis onderzoekt daarom mechanismen om informatie over het daadwerkelijke gebruik van de data terug te voeren naar het analytische systeem om de uitvoering van query's te versnellen.

C2: Het uitvoeren van verkennende analyses van live en historische multidimensionale data is met name veeleisend voor dataverwerkingssystemen vanwege de steeds groter wordende hoeveelheid data en de complexiteit van de query's die nodig zijn om de verkenningsdoelen te bereiken. Een groot deel van de contributie gepresenteerd in deze thesis is gericht op het identificeren van veelvoorkomende patronen in het opvragen van data bij karakteristieke verkennende acties, en op het bedenken van mechanismen voor dataverwerking die interactieve responstijden mogelijk maken voor query's die verband houden met dergelijke patronen.

C3: Traditioneel wordt het meeste rekenwerk bij de verwerking van query's uitgevoerd in backend- of servercomponenten. Als het gaat om tijdkritische applicaties die meerdere clients bedienen, kunnen de rekenkosten die aan elke individuele zoekopdracht zijn gekoppeld snel oplopen, waardoor de schaalbaarheid van de service wordt belemmerd. In deze thesis wordt een meer gebalanceerde afweging tussen server-side en client-side verwerking bestudeerd, om enerzijds de schaalbaarheid van de applicatie te bevorderen en anderzijds de mogelijkheid tot de berekening van interactieve query's te behouden.

C4: Patronen in het opvragen van data geven vaak informatiebehoeften of gebruikersvoorkeuren weer. Deze patronen zijn bijzonder dynamisch

en volatiel wanneer toepassingen met meerdere gelijktijdige clients werken. Voor dit geval geldt dat alle datastructuren die zijn gemaakt om bepaalde specifieke zoekopdrachten te versnellen, snel achterhaald zijn. Proactieve mechanismen zijn vereist om aan veranderingen in deze ophaalpatronen te voldoen en om te anticiperen op requests van clients, om zodoende query's met lage responstijd in tijdkritische toepassingen te kunnen garanderen.

Elk van de voorgaande uitdagingen wordt onderzocht in de hoofdstukken waaruit deze thesis bestaat.

De eerste van de genoemde uitdagingen wordt bestudeerd in Hoofdstuk 2 en Bijlage A. In deze onderdelen wordt dieper ingegaan op een mechanisme die, rekening houdend met de workload, een keuze voor een view maakt in het geval van multidimensionale data. Het voorgestelde mechanisme is gebaseerd op het uitgangspunt dat het data schema automatisch kan worden aangepast om aan de werkelijke workload te voldoen. Gematerialiseerde views zijn redundante datastructuren die analytische systemen in staat stellen om anders tijdrovende query's onmiddellijk op te lossen door hun resultaten vooraf te berekenen en op te slaan. In die zin is de methode voor het selecteren van views die in deze thesis is geïntroduceerd, gebaseerd op syntactische analyse van de query's die een bepaalde analytische workload vormen om terugkerende structurele patronen te identificeren. Query's die veelvoorkomende patronen delen, worden samen in clusters gerangschikt. Vervolgens wordt een score toegekend aan de verkregen clusterconfiguratie en worden irrelevante clusters uitgesloten van verdere verwerking. Ten slotte worden definities van views afgeleid op basis van de query's binnen elk van de resterende clusters, en de bijbehorende resultaten worden gematerialiseerd op basis van een momentopname van de tot nu toe beschikbare data. Het voorgestelde mechanisme is ook in staat om ongeziene query's in de berekende clusterconfiguratie in te passen en deze te vertalen zodat ze worden uitgevoerd met de beschikbare gematerialiseerde views. De evaluaties uitgevoerd op een conventionele database met één instantie enerzijds, en op een gedistribueerde opslagconfiguratie anderzijds, bewijzen dat het voorgestelde mechanisme in staat is om een uitgebreide reeks gematerialiseerde views te genereren. Bovendien leiden de verkregen views tot een substantiële reductie van de uitvoeringstijd, in tegenstelling tot de performantie van query's die draaien op de originele dataset.

Hoofdstuk 3 gaat dieper in op een generiek raamwerk voor dataverwerking waarmee veelvoorkomende verkenningstaken interactief kunnen worden uitgevoerd over livestreams van multidimensionale data. De methoden die in dit kader zijn bedacht, hebben tot doel om binnen minder dan een seconde een antwoord te vinden op de query's die ten grondslag liggen aan deze verkennende taken, ongeacht of het gaat om het scannen van de volledige data

die tot nu toe zijn verzameld, of alleen een recente subset van records. Het identificeren van typische interactiepatronen die worden gebruikt door consumenten van visuele verkenningstoepassingen is essentieel bij het ontwerpen van de pijplijn voor dataverwerking waarop het voorgestelde raamwerk is gebaseerd. Een van de belangrijkste functies van deze pijplijn bestaat uit het continu berekenen van synopsis datastructuren—namelijk continue views—wanneer nieuwe data in het systeem komen. Deze structuren bieden een samengevat beeld van de originele data, hetgeen een onmiddellijk antwoord geeft op de query's die verband houden met de geïdentificeerde interactiepatronen. Om dit te bereiken wordt een afweging gemaakt tussen de responstijd enerzijds en de nauwkeurigheid van het antwoord op de query anderzijds. Dit vertaalt zich in samenvattingen van data die worden berekend over gediscretiseerde verdelingen met meerdere resoluties die alle dimensies omvatten. Deze samenvattingen zijn gerangschikt in continue views en opgeslagen in een fast lookup datastore. Evenzo definieert het raamwerk de procedures die worden gebruikt om query's op te lossen die overeenkomen met de geïdentificeerde interactiepatronen, op basis van de samenvattingen binnen de continue views. Dit hoofdstuk presenteert ook twee proof of concepts van de voorgestelde aanpak. De eerste van deze implementaties is gebaseerd op een conventionele single-node geospatiale tijdreeksdatabase, terwijl de tweede een gedistribueerde stream processing engine gebruikt. Beide implementaties zijn in staat om de data streams, die afkomstig zijn van een netwerk van mobiele sensoren die zijn ingezet in een bestaande smart city omgeving in Antwerpen (België), binnen te halen en te verwerken. De analyse van de performantie van deze implementaties onthult een afname van de verwerkingstijd van query's tot twee grootte-ordes, in vergelijking met query's die werden uitgevoerd op onbewerkte data. Als we vervolgens de bovengenoemde afweging in acht nemen, observeren we dat deze snellere responstijd ten koste gaat van minder dan 10% nauwkeurigheid in de geleverde resultaten, wat voor verschillende toepassingen kan worden beschouwd als een te overwegen optie vanwege de verbeterde gebruikerservaring.

In lijn met de derde uitdaging, bestudeert hoofdstuk 4 mechanismen om een meer evenwichtige afweging te maken tussen server-side en client-side verwerking, om de schaalbaarheid van het systeem te vergroten en toch interactieve query's mogelijk te maken. De oplossing die dit hoofdstuk introduceert, bouwt voort op het raamwerk in hoofdstuk 3, en stelt een vereenvoudigde interface voor die tot doel heeft om:

1. de complexiteit van de query's die de backend kan oplossen, te beperken, waardoor de hoeveelheid rekencapaciteit die het per verzoek toewijst, wordt verminderd,

2. het hergebruik van query's voor verschillende clients te stimuleren door de cachebaarheid van requests te verbeteren, en

3. clients in staat te stellen complexe query's op te lossen door de resultaten van meerdere eenvoudigere query's te verzamelen.

Dit hoofdstuk werkt verder op een platform dat de samenvattingen van data publiceert die continu worden berekend over een stream sensordata via een lichtgewicht interface voor *Linked Data Fragments (LDF)*. Door gebruik te maken van deze interface kunnen clients alleen individuele samenvattingen ophalen, elk gebonden aan een bepaald ruimtelijk-temporeel fragment. Om clients in staat te stellen query's over willekeurige ruimtelijke regio's of tijdsintervallen op te lossen, definieert het platform een datamodel, dat compatibel is met linked data, voor het weergeven van individuele fragmenten van samenvattingen. Dit datamodel biedt hypermedia-besturingselementen waarmee clients door de ruimtelijke en temporele dimensies kunnen navigeren en zelf de relevante samenvattingen kunnen ontdekken voor het berekenen van niet-triviale query's. Bovendien wordt verbeterd hergebruik van query's bereikt als een bijproduct van de vereenvoudiging van de interface: de samenvattingen die worden gepubliceerd als *linked data fragments* zijn onveranderlijke, oneindig cachebare datastructuren. De evaluatie die is uitgevoerd op een implementatie van het voorgestelde platform, waarbij opnieuw gebruik wordt gemaakt van bovengenoemde sensordata, toont aan dat de LDF-interface een groot deel van de queryverwerking kan verplaatsen van de server-side naar de client-side van de applicatie, dankzij de verbeterde cachebaarheid. Tegelijkertijd kunnen clients een responsieve gebruikerservaring bieden door snel incrementele antwoorden op requests te leveren, terwijl de relevante fragmenten van samenvattingen worden opgehaald en verwerkt.

Hoofdstuk 5 behandelt de laatste van de hierboven genoemde uitdagingen. Om het systeem aan te passen aan verschillende patronen voor het ophalen van data die ontstaan door meerdere gelijktijdige clients die een tijdkritische service gebruiken, stelt dit hoofdstuk een netwerk-ondersteund mechanisme voor voorverwerking van query's voor, geïnspireerd op het framework dat is besproken in hoofdstuk 3. Het mechanisme wordt gepresenteerd in de context van tile-based immersieve videostreamingdiensten. Bij dit soort diensten wordt inhoud (d.w.z. data) gedefinieerd in termen van verschillende dimensies, waaronder ruimte (video tiles), tijd (videosegmenten) en videokwaliteit (tiles worden weergegeven in verschillende kwaliteitsweergaven). De Quality of Experience (QoE) van de gebruikers wordt sterk beïnvloed door de hoeveelheid data die per tijdseenheid het apparaat van de client kan bereiken, een metriek die op zijn beurt wordt beïnvloed door de vertraging die het publiek netwerk tussen client en server introduceert. Bij het bedienen van meerdere gelijktijdige clients die dezelfde video-inhoud binnen een smal tijdvenster bekijken (bijvoorbeeld on-demand videostreaming), is het vaak het geval dat kijkers zich concentreren op bepaalde specifieke delen van het scherm. Het voorgestelde mechanisme heeft tot doel dit gedeelde *Field*

of View (FoV) per videosegment te voorspellen en de bijbehorende video tiles vooraf op te halen in een edge server die caching ondersteunt. Op deze manier wordt relevante video-inhoud beschikbaar gesteld aan clients vanaf een server die geografisch zo dicht mogelijk geplaatst is, waardoor de netwerkmetrieken aanzienlijk verbeteren. Een evaluatie van het voorgestelde mechanisme heeft uitgewezen dat het een verbeterde QoE kan leveren voor clients van deze diensten. Met behulp van viewport traces die zijn verzameld tijdens uitgevoerde 360°-videostreamingsessies, werd bewezen dat het mechanisme voor prefetching in staat is om een hogere videokwaliteit te bieden zonder bevriezingen van het beeld, in vergelijking met zowel een conventionele client-serverconfiguratie, als een implementatie van een traditionele *least recently used* (LRU) cachestrategie. Bovendien wordt door het vooraf ophalen van de inhoud die kijkers eerder zullen consumeren, de belasting van de streamingserver en het verkeer op de backhaul aanzienlijk verminderd.

De verschillende frameworks en mechanismen die in deze thesis worden voorgesteld, dragen bij aan het aanpakken van de genoemde uitdagingen met betrekking tot het realiseren van interactieve query's op big data. Op basis van het resultaat van dit doctoraatsonderzoek worden interessante mogelijkheden voor verder onderzoek geïdentificeerd en besproken in hoofdstuk 6.

Summary

The rapid commoditization of data storage, computing power, and network capacity over the last decades has led to an explosive growth of the volume of data available, setting a trend that is expected to accelerate in the coming years. According to a recent forecast by the International Data Corporation (IDC), the amount of data generated worldwide only in 2021 is set to reach 79 zettabytes (i.e., 79×10^{21} bytes). Furthermore, this metric will compound over the next five years, placing the volume of data created globally at 181 zettabytes by 2025. In the modern hyperconnected world, virtually every interaction we have with our surroundings leaves behind a digital footprint. Organizations in all sectors are increasingly turning to Internet of Things (IoT) technologies to monitor their operations and collect data concerning their business activity. As the volume, variety, and complexity of data grow larger, so does the difficulty for processing, analyzing, and distilling insights from it. Such *big data* sets have long outgrown the capabilities offered by traditional data management technologies, and have brought to the forefront the need for alternative mechanisms for persisting and accessing data in a reliable, scalable and performant manner. Research and innovation in big data management have led to the development of a vast variety of solutions each one catering to certain particular use cases. These big data technologies range from NoSQL databases able to accommodate data encoded in a diversity of data models (e.g., document-based, key-value data, graph, and columnar-based), to distributed systems capable of harnessing the resources of a cluster of commodity machines to store and run data-intensive workloads.

For modern organizations, being able to access data and derive insights from it as soon as it is generated has become of utmost importance to support opportune decision-making. The need to minimize this *data-to-insight* time has led to an increasing interest in *event-based architectures* and *near real-time stream processing frameworks*. However, in spite of their growing popularity, these kinds of solutions remain largely untapped by businesses. Most of the data processing conducted nowadays still predominantly relies on batch processing methods which are known to be subject to high latency. In such circumstances, organizations face the risk of making decisions and taking action on stale data, especially for latency-sensitive applications running on these large, high-dimensional data collections. This dissertation sets

off to advance in the understanding of the problem of enabling interactive low-latency querying on large multidimensional data sets. In relation to this problem, four major challenges are addressed throughout the dissertation:

C1: Dimensional modeling is a staple technique for structuring historical business data, which resorts to denormalized schemas to enable fast processing of analytical queries. However, once defined, these schemas offer little flexibility to adapt to changes in the workload and to optimize for time-consuming queries. This dissertation explores mechanisms for feeding information about the actual use of the data back to the analytical system to speed up query execution.

C2: Performing exploratory analysis over live and historical multidimensional data is particularly demanding for data processing systems due to the ever-growing volume of data and the complexity of the queries required to fulfill the exploration goals. A large part of the contributions presented in this thesis are aimed at identifying common data access patterns behind typical exploratory actions, and at devising data processing mechanisms enabling interactive response times for queries associated with said patterns.

C3: Traditionally, most of the heavy lifting of query processing is conducted in backend or server-side components. When it comes to time-sensitive applications serving multiple clients, computational costs linked to each individual query can quickly add up thus hampering service scalability. A more balanced trade-off between server-side and client-side processing is studied in this dissertation, to favor system scalability while still delivering interactive query computation.

C4: Data retrieval patterns often reflect information needs or user preferences. These patterns are particularly dynamic and volatile when applications deal with multiple simultaneous clients. In these circumstances, any data structures created to speed up certain specific queries are quickly rendered obsolete. Proactive mechanisms to adjust to changes in these retrieval patterns and anticipate client requests are required to ensure low-latency querying in time-critical applications.

Each of the foregoing challenges is addressed in the research chapters composing this dissertation.

The first of the stated challenges is studied in Chapter 2 and Appendix A. These chapters elaborate on a workload-aware mechanism for view selection in large dimensional data. The proposed mechanism is based upon the premise that the data schema can be automatically adapted to meet the actual workload demands. Materialized views are redundant data structures

enabling analytical systems to instantly resolve otherwise time-consuming queries, by pre-computing and storing their results. In that sense, the view selection method introduced in this dissertation relies on syntactic analysis of the queries composing a certain analytical workload to identify recurrent structural patterns. Queries sharing common patterns are arranged together into clusters. Then, the obtained cluster configuration is scored and spurious clusters are ruled out from further processing. Finally, view definitions are derived based on the query statements within each of the remaining clusters, and their corresponding results get materialized based on a snapshot of the data available thus far. The proposed mechanism is also able to fit unseen queries into the computed cluster configuration, and translate them so that they run against the available materialized views. The experimental evaluation conducted on a conventional single-node database, as well as a distributed storage setup, proved the devised mechanism effective in generating a comprehensive set of materialized views. Moreover, the obtained views lead to a substantial reduction of the workload execution time, in contrast to the performance of queries running on the base dataset.

Chapter 3 elaborates on a generic data processing framework for enabling common exploratory tasks to run interactively over live streams of multidimensional data. The methods devised in this framework aim at achieving sub-second responses for queries underlying said exploratory tasks, regardless of whether they involve scanning the entire data collected thus far, or only a recent subset of records. Identifying typical interaction patterns adopted by consumers of visual exploratory applications is instrumental in designing the data processing pipeline this framework thrives on. One of the key functions of this pipeline consists in continuously computing synopsis data structures—namely, *continuous views*—as new data comes into the system. Said structures represent a summarized view of the original data, which enables instantaneous resolution of the queries associated with the identified interaction patterns. To achieve this, a trade-off between query response time and accuracy is proposed, which translates into data summaries being computed across discretized bins of multiple resolutions spanning all data dimensions. These data summaries are arranged into continuous views and stored in a fast lookup datastore. Likewise, the framework defines the formal procedures used to resolve queries matching the identified interaction patterns, based on the summary data within the continuous views. This chapter also presents two proof of concepts of the devised approach using smart city sensor data as a relevant use-case. The first of these implementations is based on a conventional single-node geospatial time-series database, while the second one uses a distributed stream processing engine. Both implementations are able to ingest and process the stream of observations coming from a network of mobile sensors deployed within a real smart city environment in Antwerp, Belgium. A performance evaluation conducted on the implementations of the proposed framework revealed a decrease in query

processing time of up to two orders of magnitude, in comparison to queries running against the base raw data. By virtue of the above-mentioned trade-off, this speed-up in query response time comes at the expense of less than 10% accuracy in the delivered results, which for several applications can be regarded as a reasonable price to pay in exchange for an enhanced user experience.

In line with the third stated challenge, Chapter 4 studies mechanisms to achieve a more balanced trade-off between server-side and client-side processing, to boost system scalability while still offering interactive querying performance. The approach this chapter introduces builds upon the framework in chapter 3, and proposes a simplified querying interface that aims at:

1. limiting the complexity of the queries that the backend application is able to resolve, thus reducing the amount of computing resources it allocates per request,

2. boosting query reuse across several clients by improving request cacheability, and

3. enabling clients to solve complex queries by collating the results of multiple simpler ones.

This chapter elaborates on a platform that publishes the data summaries being continuously computed over a stream of sensor data via a lightweight *Linked Data Fragments* (LDF) interface. By using this interface clients are only allowed to retrieve individual data summaries, each one bound to a certain spatio-temporal fragment. To enable clients to resolve queries over arbitrary spatial regions or time intervals, the platform defines a linked-data compliant data model for representing individual summary fragments. This data model provides hypermedia controls that allow clients to navigate across the spatial and temporal data dimensions, and discover by themselves the relevant summaries for computing non-trivial queries. Furthermore, enhanced query reuse is achieved as a by-product of simplifying the querying interface: the data summaries published as linked-data fragments are immutable, infinitely cacheable data structures. The experimental evaluation conducted on an implementation of the proposed platform using sensor data, shows that the LDF interface can offload a large part of the query processing from the server to the application's client-side, thanks to the improved cacheability. At the same time, clients can offer a responsive user experience by promptly delivering incremental answers to user requests, as the relevant summary fragments are retrieved and processed.

Chapter 5 deals with the last of the challenges listed above. To adapt to varying data retrieval patterns that emerge from multiple concurrent clients

consuming a latency-sensitive service, this chapter proposes a network-assisted query preprocessing mechanism, inspired by the framework discussed in chapter 3. The mechanism is presented in the context of tile-based immersive video streaming services. In this kind of service, content (i.e., data) is defined in terms of several dimensions including space (video tiles), time (video segments) and video quality (tiles are served in various quality representations). The user quality of experience (QoE) is heavily influenced by the network throughput perceived at the client's device, which in turn is affected by the network latency. When serving multiple concurrent clients consuming the same video content within a narrow time window (e.g, near-live on-demand video streaming), it is often the case for viewers to focus on certain common regions of the display. The proposed mechanism intends to predict this shared *field of view* (FoV) on a per video segment basis, and prefetch the corresponding video tiles into a cache-enabled edge server. In this way, relevant video content is made available to clients from a nearby location, reducing the network latency, and increasing the perceived link capacity in consequence. An evaluation conducted on the proposed mechanism evidenced that it can deliver an enhanced QoE for consumers of these services. Using viewport traces collected from actual 360° video streaming sessions, the evaluation proved the prefetching mechanism capable of serving higher video quality, and a freeze-free playback experience in comparison with both a conventional client-server setup, and an implementation of a traditional *least recently used* (LRU) cache replacement strategy. Furthermore, by prefetching the content that active viewers are more likely to consume, the load on the content server, as well as the traffic on the backhaul network are substantially reduced.

The frameworks and mechanisms proposed in this dissertation contribute to address the stated challenges related to achieving interactive querying performance on big datasets. As a result of this study, interesting venues for further research have been identified and discussed in Chapter 6.

1 Introduction

"We go back and forth between being time's master and its victim."

—James Gleick (1954–)

1.1 A brief overview of Big Data

A recent report on digital trends found that the global number of internet users reached more than 4.7 billion, and mobile phone users around 5.3 billion by April 2021 [1]. This is at least 60% and 67% of the total world population, respectively. As computer technology and telecommunication networks grow increasingly more advanced and capable, their access barriers continue to decline. Nowadays, it is common for us to walk around with a smart device in our pockets, whose computing power and storage capacity exceed by several orders of magnitude those from the *guidance computer* aboard the Apollo 11 spacecraft that took the man to the moon for the first time in 1969 [2, 3]. This commoditization of computing and connectivity has boosted our ability to generate and store data. In the hyperconnected world we live in, nearly every action we take leaves behind a digital trail. From the mundane such as the number of steps we walk in a day, or the posts we like in social media, to the critical such as medical records, tax returns or bank account details, etc. Moreover, a large number of devices we interact with on a daily basis—office and household appliances, the vehicles we use to commute, and large industrial equipment—is today connected to

the Internet generating endless streams of data. The Internet of Things (IoT), as this growing network of machines is known, is estimated to have reached about 12 billion connected devices in 2020 [4], and is responsible for over 40% of the internet data generated nowadays [5]. With both non-IoT and IoT networks in continuous expansion, our digital data universe is growing at an unprecedented accelerated rate.

Back in the mid-nineties, amidst the early years of the World Wide Web (WWW) boom, computer scientists like John Mashely and Sholom M. Weiss were already anticipating the current explosive growth in available data [6, 7]. Since then the term *big data* started to be adopted to refer to this emerging phenomenon. These early visionaries realized the big data's disruptive potential in supporting decision-making processes and building more accurate forecast models. This notion of big data grew increasingly popular as the Web transitioned from a network of linked documents mainly based on text, to a global-reaching multimedia exchange platform. The world witnessed the dawn of several of today's most valuable technology companies, such as Amazon, Google and Facebook, among others. These organizations transformed industries in all sectors by harnessing the ubiquity and openness of the Web to collect and analyze massive amounts of data from their users. As profitable as data harvesting proved to be for these companies, big data management came with daunting technical challenges. For one thing, besides the sheer volume of data, there is the issue of variety. In contrast to traditional relational databases, in which data entities are strictly defined under a normalized schema, big data systems deal with semi-structured and unstructured data served from a myriad of possible sources. Furthermore, big datasets are often subject to variability, which refers to changes in the data structures, content, and formats they adopt over time. Lastly, big data systems typically incorporate data sources delivering continuous streams of data at high rates (e.g., IoT setups, financial services, Web applications). Adding to the volume, this high-velocity data drives the need for partitioning and parallelism of storage and processing, which can potentially harm the consistency and availability of these big data collections [8].

The above-mentioned features, often referred to as the *big data Vs* (*Volume*, *Variety*, *Variability* and *Velocity*) [9] , brought to the forefront the need for new data management strategies that enable large-scale, reliable storage and processing of complex, high-dimensional data. One of the most representative technologies developed as a response to this need is the *Hadoop Framework* [10]. Based on the *Google File System* paper by Ghemawat et al. [11], Hadoop adopts a shared-nothing architecture that allows for distributed storage and processing of vast amounts of data on a cluster of commodity hardware. Hadoop is built on two basic components: (1) the

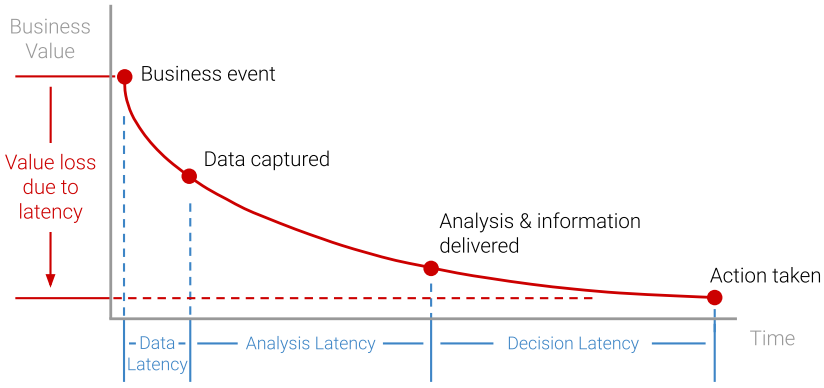
Hadoop Distributed File System (HDFS) which offers high availability and fault tolerance via data replication; and (2) the *MapReduce framework* [12], which provides a straightforward programming model that enables the parallel execution of batch jobs across the nodes of the Hadoop cluster. For a long time, Hadoop was the premier open-source platform for big data processing. At the time of this writing, every major cloud computing provider offers a managed distribution of the Hadoop framework as part of their service portfolio. Along with other *NoSQL* data technologies such as MongoDB [13] and Cassandra [14] also popularized by Web-scale platforms, Hadoop laid the foundation for the modern big data technology landscape.
The relentless innovation within the industry and a thriving open source community, brought about tools such as Apache Spark [15] and Apache Tez [16] which offer a more time-efficient alternative to Hadoop's MapReduce framework and (in Spark's specific case) support for processing data in motion via a streaming API. Today, organizations benefit from a burgeoning big data ecosystem providing tools that cater to particular use cases: from high throughput in-memory key-value stores [17], time-series databases [18, 19], and message brokers supporting event-driven applications [20], to ACID-compliant distributed databases [21, 22], and Multi-model data stores [23]. The data analysis capabilities these big data technologies enable have proved valuable in several critical scenarios. For instance, accurate statistical models on aggregated data collected from telecom operators, social networks, online news stories, among other sources, helped in the contention of the Ebola outbreak in West Africa back in 2014 [24]. More recently, during the COVID-19 global health crisis, big data analysis has been crucial for governments around the world to understand how their measures are affecting the spread of the virus, allowing them to adjust their course of action accordingly [25, 26]. Even though big data has become a commonplace asset for organizations in all sectors, it is still a nascent research field with countless open challenges and opportunities for innovation.

1.2 Problem Statement

Data technologies are at the forefront of the digital transformation businesses are currently undergoing. As these businesses turn into data-driven organizations, reducing the time-to-insight—i.e., the average time it takes to produce actionable insight from the moment at which data is generated—becomes increasingly important for them to ensure optimal decision-making and timely strategic actions. In data-intensive applications, the perceived value of data heavily depends on how fast it is captured, processed, and analyzed (see Figure 1.1). This is certainly the case for numerous application

domains such as IoT, supply chain, fraud detection, and networking and security, in which prompt response to potentially hazardous events is required to prevent or mitigate their adverse effects. When it comes to big data analysis, it is often the case that business questions can only be answered by running analytical workloads over a large volume of data that spans multiple silos or sources. In these circumstances, the processing required (breaking silos, joining multi-dimensional datasets, computing aggregates, etc) is usually resource-intensive and time-consuming. In consequence, organizations run the risk of making decisions on stale data.

Figure 1.1 Data value from capture to business action. Adapted from [27]



Under these conditions, informed and timely decision making requires mechanisms that enable high throughput data processing and efficient querying. This while dealing with constraints of consistency, availability and partition tolerance inherent to distributed computing systems [28]. In spite of the clear need for mechanisms and tools supporting interactive querying of continuous generated data, this remains a largely open big data problem in which three main requirements are to be met, namely: (*i*) high throughput resolution of arbitrary queries, enabling data analysis and information retrieval on an as-needed basis, (*ii*) deal with heterogeneous data sources, since data is regularly available in a myriad of structured and unstructured formats, and (*iii*) strict latency demands. There exist some approaches conceived to tackle this problem, ranging from *Massive Parallel Processing* (MPP) query engines running on top of frameworks like Hadoop (e.g., Presto [29], Apache Impala [30], Apache Drill [31]), to technology-agnostic architecture patterns such as the Lambda [8] and Kappa [32] Architectures. However, these solutions lack any guidelines concerning how to ensure interactive response times for time-critical applications when querying large, multi-dimensional datasets. This dissertation documents a study around

this problem and proposes various workload-aware processing mechanisms that aim at optimizing the query response times of time-sensitive data applications, across multiple application domains. It is worth noting that, in the context of the research documented herein, the terms *interactive querying* and *low-latency querying* are used interchangeably to denote the need for prompt feedback to the questions users pose to data in time-critical domains. Similarly, the term *data dimension* refers to the different aspects of data (e.g., time, location, sensed variables, etc.) that may be relevant to the users' queries. That said, the research questions that shaped the work in this dissertation are discussed below:

1. How can the structure of a large dimensional dataset be dynamically adjusted to ensure instantaneous resolution of recurrent analytical queries?

Dimensional modeling is one of the foundational techniques for laying out the structure of historical business data in modern enterprise analytical systems. It consists in arranging data into denormalized schemas in which qualitative, seldom-changing elements describing business events are factored out from the most dynamic quantitative information concerning business performance (e.g., purchase and sale orders, expenses). The former descriptive elements are persisted into data entities known as *dimension tables*, while performance-related data is stored into structures called *fact tables*. In this way, each record within a fact table is described via its relationship to each of the dimension tables. Analytical queries issued against dimensionally modeled datasets can be expensive and time-consuming depending, among other factors, on the number of join operations they involve, the cardinality of the fields included in the projection clause, and the aggregate operations applied. By keeping a record of the queries submitted to the dataset, it is possible to identify recurrent access patterns which afterwards might be used to further denormalize the given dimensional schema (e.g., merging dimension tables, partitioning fact tables), or creating redundant data structures such as *data views*, so that queries complying to those patterns can be resolved much faster. This dissertation explores how information about the actual use of the data can be fed back into the analytical system to speed up query execution.

2. How can common exploratory analysis tasks be supported on live streams of multidimensional data under interactive (low-latency) time constraints?

While Research Question 1 pondered over mechanisms for ensuring low-

latency querying on a large but temporary immutable dataset, in this second question the latter assumption is relaxed to tackle the more general case in which records are continuously appended to a constantly growing dataset. Once again, the effective use of the data is considered to shape the scope of the research in this direction. In this sense, common data access patterns should be identified as a starting point, and automatic mechanisms optimizing query performance for said patterns should be devised. A second major aspect this research question focuses on is interactivity. For time-critical data applications, ensuring prompt query resolution is crucial, not only to prevent users from making decisions on stale data, but also to enable a smooth and responsive user experience. Meeting such stringent latency requirements entails trade-offs between different aspects of the system such as accuracy, flexibility of the querying interface, and query processing time. These trade-offs are investigated in this dissertation as well.

3. How can a balanced trade-off between server and client-side computation be reached to enable interactive exploration applications to serve requests from a large number of users?

Server-side processing can be cost-prohibitive for data exploration applications serving multiple clients under interactive time constraints, let alone when operating at Web-scale. By simplifying the querying interface, it is possible to offload the backend servers favoring the system's scalability. Such a lightweight interface should provide elemental, cacheable answers which client-side applications can collate afterward to resolve complex queries by themselves. Existing Web technologies (*knowledge graphs*, *Linked Data*, and *HTTP caching*) provide means to define such cost-effective data interfaces allowing to transfer part of the heavy lifting of the query computation from the server to the clients. The trade-off between client and server-side processing needs to be investigated to aid system scalability while still providing low-latency request resolution and an interactive user experience.

4. How can data retrieval patterns be identified in a streaming setting to dynamically enhance the user experience of multiple concurrent clients served by a latency-sensitive application?

For real-world production applications, it is often the case to have multiple clients consume the same data content within a small time frame. Shared query patterns can be uncovered by analyzing how clients retrieve data over a given period of time, which allows precomputing the result of recurrent requests and storing them into a fast-lookup data structure for later retrieval. However, these patterns can be highly volatile and change over time as

clients come in and out of the system and user interests evolve. This dissertation investigates how stream processing can help in identifying these recurrent patterns and dynamically adjust them over time to accurately reflect clients' consumption preferences.

1.3 Research Contributions

The research questions stated in the previous section provide the framework within which the work presented in this dissertation was conducted. The main research contributions made in line with each ofthese questions are summarized in the paragraphs below:

1. A workload-aware method for automatic view selection on large dimensional datasets, intended to enable interactive-level latency for both data exploration and visualization tasks.

Aligned with *Research Question 1*, this view selection method is a realization of a conceptual framework—also devised within the scope of the research documented in this dissertation—which contemplates a systematic mechanism for incremental optimization of the schema of a dimensional dataset. The optimization proposed is based on detecting recurrent patterns in the structure of time-consuming analytical queries issued against said dataset over time. The rationale behind this is to apply a series of operations (e.g., view materialization, table partitioning, field indexing) that result in a rearranged schema, against which queries satisfying the identified patterns can be instantly resolved.
The view selection method in particular, relies on syntactic analysis of the query statements composing a given analytical workload. For this, a query representation was first devised which encodes not only the entity-attribute usage of each of the statements within the workload, but also the structure of conventional analytical queries. This equivalent vectorized form is fed to a clustering algorithm from which a limited set of candidate views is derived, based on a custom query dissimilarity function. Finally, the decision on which views to materialize is made according to how well their corresponding clusters fare in terms of a *materializable score*. This score accounts for both *cluster consistency* (i.e., how similar are the queries inside a given cluster, and how distant are they from queries in other clusters) and *cluster size* (the larger the cluster in relation to the workload size the better). The proposed method was validated on the widely adopted *Star Schema Benchmark* [33] using a conventional single-node database setup and a distributed storage setting. The performance evaluation proved the devised mechanism effective

in deriving a consistent set of materialized views that lead to a significant reduction of the query response time, compared to the system performance observed on the base dataset.

2. A generic framework for serving interactive low-latency requests, typical of visual exploratory applications on live data streams.

The solution proposed in *Research Contribution 1* assumes an immutable, size-bounded dataset. Moreover, the formulated view selection mechanism is applied conforming to a batch processing strategy. In contrast, the framework referred to in this second contribution deals with the problem of ensuring low-latency querying on an ever-growing dataset fed by a continuous data stream. The intent behind this framework is to enable interactive data exploration over the entire history of the dataset, including the most recent entries. In this sense, the starting point for formulating the framework is the identification of the most common interaction patterns adopted by users of visual exploratory applications on spatio-temporal data. Then, a categorization of the queries satisfying the information needs associated to each of these patterns is conducted. To speed up the execution of queries fitting the derived categories, the framework incorporates a technology-agnostic data processing pipeline which allows for the continuous estimation of data synopsis structures over the stream of observations. These synopsis structures are computed across a spatio-temporal fragmentation scheme, spanning multiple resolutions in all data dimensions. These structures make for a dynamically compacted dataset, able to provide instantaneous answers to queries supporting interactive exploration tasks at the expense of some accuracy. Two proof of concept implementations of the proposed framework—running on one year worth of multi-sensor data sourced from a real-world smart city environment—prove it effectively delivers sub-second execution of queries supporting common visual exploratory analysis tasks.

3. The definition of a *Linked Data Fragments* interface that enables (*1*) scalable, cost-efficient publication of live summaries of spatio-temporal data streams and (*2*) responsive, incremental query resolution for multiple concurrent clients.

This contribution answers *Research Question 3* in which the case for a more balanced trade-off between server-side and client-side computation is presented to offload the application backend and boost its ability to serve a large number of clients. The approach adopted to achieve said trade-off builds upon the framework discussed earlier for *Research Contribution 2*. In

this sense, the data summaries being continuously computed as new data comes in are arranged into discretized connected fragments, represented by a *Linked-Data* compliant data model. Each one of these fragments accounts for the temporal state of the underlying data across a discretized area, and is connected to other fragments via hypermedia controls that reflect the inherent order of the time series. The querying interface is limited to serving these individual fragments instead of answering requests over arbitrary regions and/or time intervals. In this way, it is up to the client-side application to retrieve the fragments required to resolve these complex queries (harnessing the hypermedia controls embedded in the data summaries representation), and combining them to deliver a final answer to the user. This lightweight querying interface comes with two substantial benefits: (*i*) it encourages reuse across multiple clients since fragments are immutable and infinitely cacheable structures, thus lightening the load on the system's backend, and (*ii*) client-side applications can display requests' answers in an incremental manner as new fragments are retrieved and processed, making for a responsive and interactive user experience.

4. An edge-assisted query processing method capable of anticipating the requests of multiple concurrent clients.

In time-sensitive applications serving multiple clients, data access patterns tied to user interests are often dynamic and tend to change over time. Methods that thrive on continuous query precomputation to boost read performance should adapt to this variable behavior to consistently provide a responsive user experience. Adaptive tile-based video streaming services deliver immersive content to viewer's head-mounted displays (HMD) using an equirectangular projection of the 360° video, fragmented into spatial tiles and temporal segments. To ensure an efficient use of the network bandwidth, only the tiles overlapping with the user's field of view (FoV) should be served and their quality level adjusted to the client's perceived network capacity, using HTTP adaptive streaming techniques (HAS). To prevent playout freezes due to buffer starvation, the client application should be able to anticipate viewer's head movements and request the video tiles she is likely to watch in the upcoming segments (i.e., predicted viewport). For this kind of video streaming service, network latency due to distant content servers can substantially degrade the network throughput perceived at the client-side and, in consequence, the overall user's quality of experience (QoE). Inspired by the framework presented in *Research Contribution 2*, this dissertation introduces a network-supported mechanism that essentially assembles a collective playout buffer to serve active watching sessions from a

nearby edge server. To build this collective buffer the mechanism incrementally combines the predicted viewports of concurrent streaming sessions, and downloads the most popular tiles per segment into the memory of a cache-enabled edge server. When clients start buffering content from this relatively close location, network latency is expected to be low. This in turn increases the perceived network throughput, prompting the client to request video tiles with higher quality representations for the subsequent segments. The evaluation of the proposed mechanism on a public dataset of viewport traces, recorded from real-world 360° video streaming sessions, proves it substantially enhances the viewer's QoE while reducing the backhaul traffic and content server's load.

1.4 Dissertation Outline

Besides the current introductory chapter, the remainder of this dissertation comprises four chapters addressing each of the research contributions declared in the previous section, and two complementary appendices. The chapters correspond to publications written over the course of this Ph.D. The paragraphs below summarize the content of each of the remaining chapters (and appendices).

Chapter 2 presents the workload-aware method for automatic view selection on dimensional datasets mentioned in *Research Contribution 1*. Following a definition of the context and scope of the research, this chapter goes on to describe the architecture of a dynamic data transformation framework intended for speeding up query execution through a systematic iterative process. The proposed view selection method is introduced next as a realization of this framework. It relies on a syntactic analysis procedure applied to the query statements within a given analytical workload, comprising three major stages: (*i*) *query representation*, which involves translating raw statements into a feature vector capturing the information about the structure and entity-attribute use of each of the queries, (*ii*) *query dissimilarity estimation* which applies a custom distance function to determine to what extent two queries are related, and (*iii*) *query clustering and view materialization* which adopts an agglomerative clustering method and a cluster quality metric to derive a comprehensive set of views covering groups of related queries. An experimental evaluation of the proposed approach using the *Star Schema Benchmark* [33] is discussed also in this chapter. Results account for an effective decrease in query execution time ranging from 80% to 99%, and a storage footprint that amounts to 13% of the size of the base dataset. The results presented in this chapter correspond to a proof of

concept implementation running on top of a relational database. Appendix A presents an implementation of the proposed view selection mechanism running on a distributed storage environment.

Chapter 3 introduces EXPLORA (**E**fficient e**XPLOR**ation through **A**ggregation), a framework for speeding up spatio-temporal queries supporting visual exploration tasks on live feeds of mobile sensor data, as described back in *Research Contribution 2*. This chapter provides an extensive account of the functional requirements, enabling techniques, architecture and data processing algorithms behind the operation of the framework. It also presents a detailed description of two proof-of-concept implementations of EXPLORA, one based on a traditional spatial time-series database setup, and another adopting a distributed stream processing approach. These implementations were tested on real-world sensor data collected from a smart city environment currently operating in Antwerp, Belgium [34]. An evaluation of the performance of these implementations is also discussed at the end of this chapter. The results obtained account for a decrease of up to two orders of magnitude in query processing time, compared to the performance of queries running on the raw sensor observations. In this way, this chapter shows how implementing EXPLORA can effectively lead to a substantial reduction in query response time, enabling an interactive user experience for common visual exploratory tasks over live smart city data.

Chapter 4 builds upon the framework introduced in the preceding chapter, and elaborates on EXPLORA-LD (i.e., EXPLORA *Linked Data*), a platform intended to aid server scalability in systems serving live spatio-temporal sensor data, while providing interactive querying performance. EXPLORA-LD implements the proactive approach to data ingestion detailed in the chapter about EXPLORA, by which it takes in a stream of sensor observations and continuously computes data summaries from the measured variables across the temporal and spatial dimensions. As highlighted earlier in *Research Contribution 3*, EXPLORA-LD publishes these data summaries through a lightweight, cost-efficient Linked Data Fragments (LDF) interface that offloads a large part of the query processing from the server-side to the client application. This chapter explains the design decisions behind the data model and architecture of the platform, and also compares its performance with that of an EXPLORA implementation exposing a conventional querying interface. This benchmarking evaluation is conducted on data gathered from the smart city setup referred to earlier, using the list of visual queries specified in Appendix B. The results evidence the performance gain in terms of application responsiveness and use of computing resources, thanks to the incremental querying and inherent cacheability featured by the LDF interface.

Chapter 5 delves into the details behind EXPLORA-VR (i.e., EXPLORA *Virtual Reality*), the edge-assisted mechanism for preemptive retrieval of immersive video content outlined back in *Research Contribution 4*. This chapter elaborates on the data structures that underlie the collective play-out buffer designed to serve multiple concurrent watching sessions, along with the formal definition of the stream processing procedure responsible for incrementally building it. Then, the description of the system's architecture and implementation is provided. The chapter goes on to detail the experimental setup used to simulate multiple viewers consuming tile-based 360° video content in an on-demand near-live scenario, under diverse network conditions. The proposed mechanism is tested against a conventional client-server setup, and an edge-assisted strategy based on a traditional *least recently used* (LRU) cache replacement policy. The results of this evaluation show that EXPLORA-VR not only delivers an enhanced watching experience in terms of quality and freeze-free video playback, but also substantially offloads the content server and reduces the network traffic towards the back-end by consistently serving more than 98% of the client requests from a cache-enabled edge server.
Finally, Chapter 6 concludes this dissertation summarizing the main findings of the research and highlighting various open challenges raised by this work.

1.5 Publications

The research conducted during this PhD research led to several publications in scientific journals and international conferences and workshops. The following list provides an overview of these publications.

1.5.1 Publications in International Journals

[1] **L. Ordonez-Ante**, G. Van Seghbroeck, T. Wauters, B. Volckaert and F. De Turck. *A Workload-Driven Approach for View Selection in Large Dimensional Datasets.* Journal of Network and Systems Management, vol. 28, no. 4, p. 1161-1186, 2020.

[2] **L. Ordonez-Ante**, G. Van Seghbroeck, T. Wauters, B. Volckaert and F. De Turck. EXPLORA*: Interactive Querying of Multidimensional Data in the Context of Smart Cities.* Published as special issue paper in the MDPI Sensors Journal, vol. 20, no. 9, e2737, 2020.

[3] **L. Ordonez-Ante**, S. Vermandere, B. Van de Vyvere, P. Colpaert, G. Van Seghbroeck, T. Wauters, B. Volckaert and F. De Turck. EXPLORA-LD*: a Linked Data Fragments approach for interactive*

querying on mobile sensor data in Smart Cities. Submitted to the IEEE Internet of Things Journal, January 2021.

[4] **L. Ordonez-Ante**, J. van der Hooft, T. Wauters, G. Van Seghbroeck, B. Volckaert and F. De Turck. Explora-VR*: Content Prefetching for Tile-based Immersive Video Streaming Applications.* Accepted for publication in the Journal of Network and Systems Management, February 2022.

1.5.2 Publications in International Conferences

[1] **L. Ordonez-Ante**, T. Vanhove, G. Van Seghbroeck, T. Wauters and F. De Turck. *Interactive Querying and Data Visualization for Abuse Detection in Social Network Sites.* Presented at 11th International Conference for Internet Technology and Secured Transactions (ICITST 2016).

[2] **L. Ordonez-Ante**, T. Vanhove, G. Van Seghbroeck, T. Wauters, B. Volckaert and F. De Turck. *Dynamic data transformation for low latency querying in big data systems.* Presented at 2017 IEEE International Conference on Big Data (Big Data 2017).

[3] **L. Ordonez-Ante**, G. Van Seghbroeck, T. Wauters, B. Volckaert and F. De Turck. *Automatic view selection for distributed dimensional data.* Presented at the 4th International Conference on Internet of Things, Big Data and Security (IoTBDS 2019).

1.6 Code Repositories

We advocate for open and reproducible research. The following list provides an overview of all public code repositories, generated by the candidate during his PhD research period.

[1] Explora Framework
URL: `https://github.com/IBCNServices/explora-kafka`.

[2] Explora-LD
URL: `https://github.com/IBCNServices/explora-ld`.

[3] Explora-VR Server
URL: `https://github.com/LeandroOrdonez/explora-vr-server`.

[4] Explora-VR Cache
URL: `https://github.com/LeandroOrdonez/explora-vr-cache`.

[5] EXPLORA-VR Client
URL: `https://github.com/LeandroOrdonez/explora-vr-dash-client`.

[6] Workload-driven View Selection System
URL: `https://github.ugent.be/lordezan/query_analysis`.

References

[1] Simon Kemp. Digital 2021 April Statshot Report — DataReportal – Global Digital Insights, 2021. URL https://datareportal.com/reports/digital-2021-april-global-statshot.

[2] Graham Kendall. Apollo 11 anniversary: Could an iPhone fly me to the moon?, 2019. URL https://datareportal.com/reports/digital-2021-april-global-statshot.

[3] Nick Routley. Visualizing the Trillion-Fold Increase in Computing Power, 2017. URL https://www.visualcapitalist.com/visualizing-trillion-fold-increase-computing-power/.

[4] Knud Lasse-Lueth. State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time, 2020. URL https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[5] Pethuru Raj and Ganesh Chandra Deka. *Handbook of Research on Cloud Infrastructures for Big Data Analytics*. IGI Global, USA, 1st edition, 2014. ISBN 1466658649.

[6] Francis X Diebold. On the Origin (s) and Development of the Term'Big Data'. 2012.

[7] Sholom M Weiss and Nitin Indurkhya. *Predictive data mining: a practical guide*. Morgan Kaufmann, 1998.

[8] Nathan Marz and James Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., USA, 1st edition, 2015. ISBN 1617290343.

[9] Wo L Chang and Nancy Grady. NIST Big Data Interoperability Framework: Volume 1, Definitions. 2019.

[10] Apache Software Foundation. Apache Hadoop, 2021. URL https://hadoop.apache.org/.

[11] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

[12] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] MongoDB Inc. MongoDB: The most popular database for modern apps, 2021. URL https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[14] Apache Software Foundation. Apache Cassandra, 2021. URL https://cassandra.apache.org/_/index.html.

[15] Apache Software Foundation. Apache Spark – Unified Analytics Engine for Big Data., 2021. URL https://spark.apache.org/.

[16] Apache Software Foundation. Apache Tez – Welcome to Apache TEZ., 2021. URL https://spark.apache.org/.

[17] Redis Labs. Redis, 2021. URL https://redis.io/.

[18] Influx Data. InfluxDB: Purpose-Built Open Source Time Series Database, 2021. URL https://www.influxdata.com/.

[19] Prometheus. Prometheus - Monitoring system & time series database, 2021. URL https://prometheus.io/.

[20] Apache Software Foundation. Apache Kafka: A Distributed Streaming Platform., 2021. URL https://kafka.apache.org/.

[21] Cockroach Labs. CockroachDB, 2021. URL https://www.cockroachlabs.com/product/.

[22] PingCAP Inc. TiDB, 2021. URL https://pingcap.com/products/tidb/.

[23] ArangoDB GmbH. ArangoDB, the multi-model database for graph and beyond., 2021. URL https://www.arangodb.com/.

[24] Gabriel J Milinovich, Ricardo J Soares Magalhães, and Wenbiao Hu. Role of big data in the early detection of Ebola and other emerging infectious diseases. *The Lancet Global Health*, 3(1):e20–e21, 2015.

[25] Marcello Marini, Ndaona Chokani, and Reza S Abhari. COVID-19 epidemic in switzerland: Growth prediction and containment strategy using artificial intelligence and big data. *MedRxiv*, 2020.

[26] Leesa Lin and Zhiyuan Hou. Combat COVID-19 with Artificial Intelligence and Big Data. *Journal of travel medicine*, 27(5):taaa080, 2020.

[27] Federico Pigni, Gabriele Piccoli, and Richard Watson. Digital data streams: Creating value from the real-time flow of big data. *California Management Review*, 58(3):5–25, 2016.

[28] Seth Gilbert and Nancy Lynch. Perspectives on the CAP Theorem. *Computer*, 45(2):30–36, 2012.

[29] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, et al. Presto: SQL on everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1802–1813. IEEE, 2019.

[30] Marcel Kornacker, Alexander Behm, Victor Bittorf, Taras Bobrovytsky, Casey Ching, Alan Choi, Justin Erickson, Martin Grund, Daniel Hecht, Matthew Jacobs, et al. Impala: A Modern, Open-Source SQL Engine for Hadoop. In *CIDR*, volume 1, page 9, 2015.

[31] Michael Hausenblas and Jacques Nadeau. Apache Drill: Interactive Ad-hoc Analysis at Scale. *Big data*, 1(2):100–104, 2013.

[32] Jay Kreps. Questioning the Lambda Architecture, 2014. URL https://www.oreilly.com/radar/questioning-the-lambda-architecture/.

[33] Patrick E O'Neil, Elizabeth J O'Neil, and Xuedong Chen. The Star Schema Benchmark (SSB), 2009. URL https://www.cs.umb.edu/~poneil/StarSchemaB.PDF. https://www.cs.umb.edu/~poneil/StarSchemaB.PDF. Last accessed: 2018-11-28.

[34] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester. City of things: An integrated and multi-technology testbed for IoT smart city experiments. In *2016 IEEE International Smart Cities Conference (ISC2)*, pages 1–8, Sep. 2016. doi: 10.1109/ISC2.2016.7580875.

2

A Workload-driven Approach for View Selection in Large Dimensional Datasets

Four research questions have driven the work presented in this PhD thesis. This chapter details the contributions of this work concerning the first research question. Herein a workload-aware mechanism for automatic view selection in large dimensional data is introduced. The proposed approach is based on the idea of augmenting the available data with precomputed views containing the answers to expensive analytical queries. Since creating views for each individual query is impractical in terms of storage and computing resources, the mechanism uses syntactical analysis techniques to identify groups of highly related queries. View definitions are built out of each query group, and their corresponding results get materialized based on a snapshot of the data available thus far. The experimental evaluation conducted on both a single node setup and a distributed data store (see Appendix A) demonstrate the ability of the proposed method to derive a comprehensive set of materialized views. Queries running against these views experienced a speed-up of 80% to 99.99% relative to their processing time measured on the base dataset.

⋆⋆⋆

L. Ordonez-Ante, G. Van Seghbroeck, T. Wauters, B. Volckaert, and F. De Turck.

Published in Journal of Network and Systems Management, March 2020.

Abstract The information explosion the world has witnessed in the last two decades has forced businesses to adopt a data-driven culture for them to be competitive. These data-driven businesses have access to countless sources of information, and face the challenge of making sense of overwhelming amounts of data in a efficient and reliable manner, which implies the execution of read-intensive operations. In the context of this challenge, a framework for the dynamic read-optimization of large dimensional datasets has been designed, and on top of it a workload-driven mechanism for automatic materialized view selection and creation has been developed. This chapter presents an extensive description of this mechanism, along with a proof-of-concept implementation of it and its corresponding performance evaluation. Results show that the proposed mechanism is able to derive a limited but comprehensive set of views leading to a drop in query latency ranging from 80% to 99.99% at the expense of 13% of the disk space used by the base dataset. This way, the devised mechanism enables speeding up query execution by building materialized views that match the actual demand of query workloads.

2.1 Introduction

Providing instant access to data is still an open problem. A significant part of the value proposition of nowadays data-driven organizations relies on their ability to gain prompt and actionable insight from business data, which poses stringent requirements on the response time of enterprise applications to support interactive querying and data visualization. To meet such requirements, currently available data technology offers a variety of solutions that ranges from high-level software architectural patterns such as the Lambda and Kappa architectures proposed by [1] and [2] respectively, to off-the-shelf/open-source solutions including in-memory computing platforms like Apache Ignite [3] and SQL-on-Hadoop frameworks such as Apache Impala and Apache Drill [4]. However, experimental evidence shows [5, 6] that said solutions either fail to provide instantaneous query answering, or effectively achieve such interactive functioning but at the expense of flexibility by relying on hard-coded information views.

Existing enterprise information applications typically separate analytic workload processing (supported by *Online Analytical Processing systems*—OLAP)

from the day-to-day *Online Transaction Processing* (OLTP). A key difference between these two types of workloads lies in the data models and structures they operate on: OLTP systems work on top of highly normalized data models, while OLAP workloads run against denormalized schemas featuring precomputed views derived from transactional business data. Results of a previous experimental study [7] evidence that using such read-optimized structures alone is not enough for analytical processing applications to meet strict response time requirements, even for small datasets.

Thanks to the wide variety of storage technologies available nowadays, more attention has been drawn towards *polystore* systems, also known as *polyglot persistence* systems. A well-thought-out polyglot persistence system is able to optimally use multiple storage technologies for managing the various types of data (with different write/access patterns) that an application requires to handle. This way, data requiring rapid access (e.g. *analytics and reporting data*) may be loaded into a columnar store, while frequently-written data (e.g. *user activity logs*) can lay on a key-value database. In this sense, [8] propose leveraging on polyglot persistence to boost the performance of legacy applications by means of a dynamic transformation process intended for translating data and queries between diverse data storage technologies.

Based on the work of Vanhove et al., a framework that serves as conceptual foundation for the mechanism this chapter reports on is presented in [7]. The intuition behind that framework was to progressively optimize the schema of a base dataset by applying a sequence of data transformation operations (e.g. view materialization, table partitioning, field indexing), in response to specific query usage patterns. The experimentation conducted in that early stage evidenced that when it comes to dimensionally modeled datasets, building materialized views is the method with the most significant impact on query performance, reducing response time by several orders of magnitude, followed by table partitioning.

In this sense, this chapter introduces an automatic view selection mechanism based on syntactic analysis of the queries running against dimensionally modeled datasets. The remainder of this chapter is structured as follows: Section 2.2 addresses the related work. Section 2.3 introduces the dynamic data transformation framework that underlies the view selection approach proposed herein. Section 2.4 focuses on the main contribution of this work and elaborates on the syntactic analysis conducted on the query statements. Section 2.5 describes the implementation of the proposed approach, while Section 2.6 discusses the experimental setup and results. Finally conclusions and pointers towards future work are provided in Section 2.7.

2.2 Related Work

Dimensional data modeling is one of the foundational techniques of modern data warehousing [9]. It has been extensively applied to a wide range of domains involving data analysis and decision support, due to its inherent ability to structure data for quick access and summary [10–12]. View materialization is a common methodology used on top of these dimensional data schemes for speeding up query execution. The associated overhead of implementing this methodology involves computational resources for creating and maintaining the views, and additional storage capacity for persisting them. In this sense, finding the sweet spot between the benefits and costs of this method is regarded by the research community as the *view selection problem.*

Extensive research has been conducted around the materialized view selection problem, as evidenced in several systematic reviews on the topic by [13], [14], and [15] to mention some of them. The approaches surveyed in these works consist in general of two main steps:

1. Find a set of candidate views for materialization based on metrics such as *query execution frequency*, *query access costs*, and *base-relation update frequencies.*

2. Create a set of views selected out of the set of candidate views under certain resource constraints such as *view maintenance costs* and *storage space limitations.*

The review elaborated by [14] groups existing approaches in three main categories: (*i*) heuristic approaches, (*ii*) randomized algorithmic approaches, and (*iii*) data mining approaches.

On the one hand, heuristic approaches use multidimensional lattice representations [16, 17], AND-OR graphs [18], or *Multiple View Processing Plan* (MVPP) graphs [19, 20] for selecting views for materialization. Issues regarding the exponential growth of the lattice structure when the number of dimensions increases, and the expensive process of graph generation for large and complex query workloads, greatly impact the scalability of these approaches and their actual implementation in consequence [14, 21].

The solution space of the view selection problem grows exponentially with the number of dimensions of the data, turning this into a NP-Hard problem. Randomized algorithmic approaches [22–28] emerged as an attempt to provide approximate optimal solutions to this problem, by using techniques such as *simulated annealing* (SA), *evolutionary algorithms* (EA) and *particle swarm optimization* (PSO). However, since these approaches use

multidimensional lattice representations, MVPP and AND-OR graphs as input data structures, they suffer the same scalability issues earlier seen in heuristic approaches [14].

On the other hand, data mining approaches are mainly workload-driven: candidate materialized views are selected and created based on the syntactic analysis of query sets representative of data warehouse usage [21, 29, 30]. Data-mining based solutions work with much simpler input data structures called *representative attribute matrices*, generated out of query workloads. These structures then configure a clustering context out of which candidate view definitions are derived. Aouiche et al. [21, 29] propose the *KEROUAC* method, which addresses view selection as the combinatorial problem of finding the optimal partition of a set of objects (i.e. queries), according to a metric of clustering quality called *New Condorcet criterion* (*NCC*) [31, 32]. In KEROUAC clustering is conducted through an iterative procedure similar to the one employed for building univariate decision trees which typically run in $O(dMNlogN)$ time [33] —with N being the number of objects, M the number of attributes, and d the number of nodes (i.e. clusters) in the tree—, and additionally involves the computation of the *NCC* metric on every iteration of the clustering process, which adds to the overall complexity of the method. Other similar data mining approaches for view selection, including the one from [30], involve identifying frequent accessed information by browsing across several intermediate and/or historical results, which is deemed to be a very costly and unscalable process [14].

In this sense, this chapter delves further into the approach introduced by Ordonez et al. [7], particularly by elaborating on an automatic mechanism for materialized view selection and creation. The mechanism presented in the following sections relies also on syntactic analysis of query workloads issued against a dimensionally modeled data collection. This mechanism uses a representative attribute matrix as input data structure, assembled as a collection of feature vectors encoding all the clauses of each individual query in the workload at hand. With this input, a strategy for selecting a limited set of candidate materializable views is implemented, comprising the use of hierarchical clustering along with a custom query distance function, and the estimation of a *materializable score* on the resulting clustering configuration.

It is noteworthy that the approach in [7] was concerned with defining a data transformation framework on a conceptual level, while the mechanism discussed herein addresses an actual realization of such framework, tackling the problem of materialized view selection on large data collections. Likewise, the approach introduced by Vanhove et al. [8] —that served as inspiration for the framework proposed in [7]— is not particularly concerned

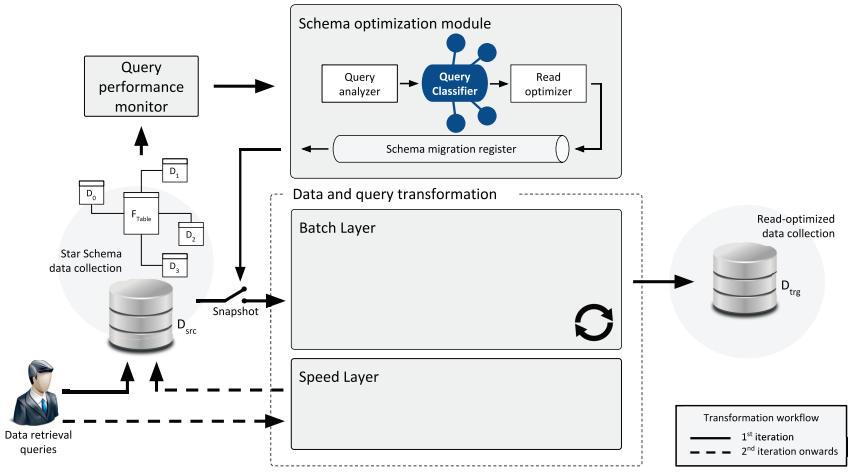
with reducing query latency, but with enabling live data migration between different data storage technologies, irrespective of the query-workload. In this sense, the contribution of the proposed mechanism lies in three key features: (*i*) a vector representation that encodes not only the query-attribute usage, but also the basic structure of analytical queries, enabling a more precise and also regular representation of the query set, (*ii*) a measure of query distance tightly suited to the structure of the formulated feature vector representation providing a more accurate method for estimating query relatedness, instead of plain Hamming distance used in existing approaches, and (*iii*) a scalable procedure for candidate view generation that relies on a measure of cluster consistency, which in turn uses the above-mentioned query dissimilarity metric to unambiguously identify *materializable* groups of related queries.

2.3 Dynamic Read-Optimization Framework

2.3.1 Iterative data transformation

The view selection mechanism this chapter presents is framed within the dynamic transformation framework introduced in a previous work [7], which aims at incrementally optimize the schema of a dimensionally modeled dataset to speed up query execution. Figure 2.1 presents an overview of the architecture of the mentioned framework, which operates by running an iterative process described in detail in [7].

Figure 2.1 Dynamic data transformation for read-optimization: architecture overview



During the first iteration of such process queries are handled by a base dataset complying a star dimensional schema (D_{src}), as their performance information is collected by the *query performance monitor*. This information is then used by the *schema optimization module* to classify incoming queries according to the read-optimization method they benefit the most, and to prescribe a *schema migration specification* defining a number of actions to be applied to a snapshot of D_{src}, to ultimately generate a read-optimized version of said dataset (D_{trg} in Fig. 2.1).
In subsequent iterations, the just generated D_{trg} becomes the new D_{src} and the process described for the first iteration is applied, except that this time incoming queries first have to undergo a translation procedure (*speed layer* in Fig. 2.1) that adjusts the query statements to the new read-optimized schema. This way client applications can issue queries to D_{trg} as if they were querying the original D_{src}.

2.3.2 Materialized view selection

The framework proposed in [7] contemplates two categories of methods for read-optimization: (1) *redundant structures*: *view materialization* and *indexing*, and (2) *non-redundant structures*: *table partitioning (horizontal* and *vertical)*. Redundant structures involve an overhead in both storage and computation, while methods from the second category imply reshaping the dataset schema, and rearranging the original information without incurring in any storage overhead. However, *non-redundant structure* methods are mostly intended for aiding the maintenance of large datasets (e.g. loading and removing vast amounts of data), rather than improving query performance by themselves.
These methods are not competing nor mutually exclusive. Conversely, approaches such as the one presented by [34] promote the combination of *redundant* and *non-redundant structures* to attain a better performance for a given query workload. This chapter deals specifically with the design and realization of a mechanism for materialized view selection, as a first step towards the implementation of the data transformation framework depicted earlier (See Fig. 2.1). Before addressing an overview of the proposed approach, the *view selection problem* is defined next.

Definition 2.1 View selection problem. *Based on the definition by [35]: Let $\mathcal{R}$ be the set of base relations, $\mathcal{S}$ the available storage space, $\mathcal{Q}$ a workload on $\mathcal{R}$, $\mathcal{L}$ the function for estimating the cost of query processing. The view selection problem is to find the set of views $\mathcal{V}$ (view configuration) over $\mathcal{R}$ whose total size is at most $\mathcal{S}$ and that minimizes $\mathcal{L}(\mathcal{R}, \mathcal{V}, \mathcal{Q})$*

In the context of the view selection approach proposed herein, some assumptions are made for the system to identify and materialize candidate views:

1. The source data collection (D_{src}) is *temporary immutable*. This implies dealing with insert and update operations is out of the scope of the mechanism presented herein.

2. The query processor provides statistical information regarding D_{src}, such as the size (row count) of each of the base tables composing the schema of the dataset, as well as the cardinality of the attributes that make up these relations.

3. Latency is favored over view storage cost. This means that the decision on materializing candidate views is driven not by storage restrictions, but by the gain in query latency.

In terms of Definition 2.1, given a dimensionally modeled dataset $\mathcal{R}$, and a workload $\mathcal{Q}$, the view selection mechanism starts by translating the queries in $\mathcal{Q}$ into feature vectors. In contrast to similar query representation techniques [21], the method proposed in this work accounts not only for query-attribute usage, but also for query structure by defining a number of regions/segments representative of each of the clauses of a Select-Project-Join (SPJ) query, i.e. aggregate operation, projection, join predicates and range predicates. This way, the devised query representation provides a more precise specification of the query statements in $\mathcal{Q}$. A detailed description of this query representation is provided in section 2.4.1.
The collection of feature vectors of $\mathcal{Q}$ configures a clustering context $\mathcal{C}$. This context is then fed to the *read optimizer* component (see Fig. 2.1) which implements a clustering algorithm able to identify groups of related queries based on a similarity score computed via a custom query distance function, described in detail in Section 2.4.2.
Upon running the clustering job, the resulting clustering configuration $\mathcal{K}$ comprises several groups of queries the algorithm deems to be similar. The idea behind building this clustering configuration is to be able to deduce view definitions covering the queries arranged under each cluster. This way, every cluster $\mathbf{c}_i \in \mathcal{K}$ would have an associated view $V_i \in \mathcal{V}$ ($\mathcal{V}$: set of candidate views), enabled to answer the queries in $\mathbf{c}_i$. Of course it is not feasible to materialize the full set of views in $\mathcal{V}$. Consider for instance the (unlikely but possible) case in which $\mathcal{Q}$ is a collection of orthogonal queries. In such situation, there would be as many clusters in $\mathcal{K}$ as queries in $\mathcal{Q}$, and in consequence one candidate view per query to be materialized, which storage-wise is not efficient at all. Now considering a more practical case,

the clustering algorithm might come up with spurious clusters, i.e. groups of queries that are actually not that related. To identify those spurious clusters and setting them apart from those clusters that are worth materializing, a *materializable score* is defined, taking into account a measure of cluster consistency and the cluster size $|\mathbf{c}_i|$. Further details on this score and the clustering procedure are provided later in section 2.4.3.

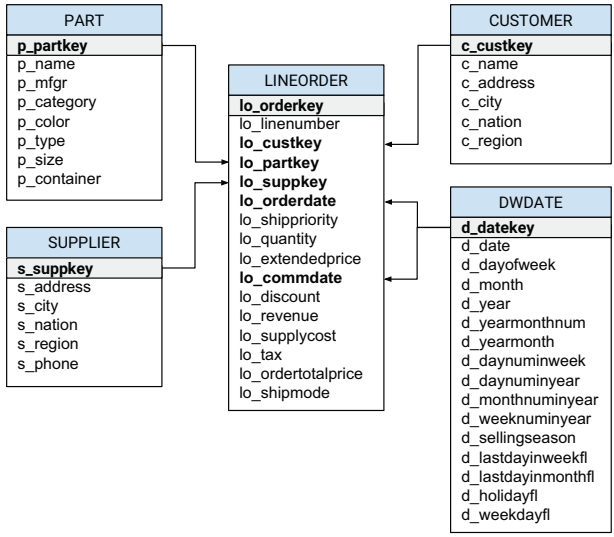
Based on the results of the *materializable score* computed on the clustering configuration $\mathcal{K}$, a subset of the candidate views in $\mathcal{V}$, $\mathcal{V}_{mat}$, is prescribed to be materialized by defining a *schema migration specification*, which is issued and executed against the source dataset (D_{src}). With the materialized views in place, the translation of the new data-retrieval queries is performed in the speed layer (See Fig. 2.1). Clearly, such translation procedure involves a certain overhead, however, according to the experimental results reported by [8], the average translation time does not exceed 100 milliseconds, which is negligible when contrasted with the query execution time on the original data collection. Finally, the translated queries run against the matching materialized views, being answered in a fraction of the time it would take to process the original queries on the base dataset.

2.4 Syntactic analysis of query statements for view selection

Since the devised view selection mechanism aims at lowering the response time of data access operations, only Select-Project-Join (SPJ) queries are considered in this analysis, particularly those containing aggregates. Aggregate-SPJ queries are one of the foundational constructs of OLAP operations. They allow for summarization returning result sets based on multiple rows grouped together under certain criteria (column projection and range predicates). In general, these aggregate queries are computationally expensive since they require scanning several records in the data collection, hence the use of materialized views for speeding up these queries.

In this research the widely-known *Star Schema Benchmark (SSB)* [36] was adopted as a baseline schema and dataset. The SSB, defines a collection of base relations along with a set of queries typically used in data warehousing. Figure 2.2 shows the entity-relationship model of SSB, featuring *Lineorder* as the table of facts, and *Customer*, *DWDate*, *Part*, and *Supplier* being the dimensions describing the facts.

Figure 2.2 The Star Schema Benchmark (SSB) data model



All statements in the SSB query set conform to the structure in listing 2.1.

```
SELECT select_list
FROM table_expression
[ WHERE search_conditions ]
[ [ GROUP BY column_list]
  [ ORDER BY column_list] ]
```

Listing 2.1: SSB query structure

The syntactic analysis this work thrives on, starts by mining the information contained in the `select_list` and `search_conditions` clauses, encoding these values in a feature vector representation that enables further query processing.

2.4.1 Query representation

The procedure for obtaining a text-mining-friendly representation of the queries takes each one of the `SELECT` statements from a workload $\mathcal{Q}$ and extracts the *aggregate* (agg_q) and *projection* ($proj_q$) elements, and *join* ($join_q$) and *range* ($rnge_q$) predicates, resulting in the following tuple:

$$q = (aggr_q, proj_q, join_q, rnge_q) \tag{2.1}$$

The tuple above is the high-level vector representation of the queries from $\mathcal{Q}$. Consider for example the following `SELECT` statement:

```
SELECT SUM(lo_revenue), d_year, p_category
FROM lineorder, dwdate, part
WHERE lo_orderdate = d_datekey
    AND lo_partkey = p_partkey
    AND d_year > 2010
GROUP BY d_year, p_category
```

For the query above:

$$
\begin{aligned}
aggr_q &= [\texttt{SUM}, \texttt{lo\_revenue}] \\
proj_q &= [\texttt{d\_year}, \texttt{p\_category}] \\
join_q &= [\texttt{d\_datekey}, \texttt{p\_partkey}] \\
rnge_q &= [\texttt{d\_year}]
\end{aligned}
$$

Each element of the above high-level vector representation gets mapped to a vector using a binary encoding function, as described below.

Definition 2.2 Binary mapping function. *Let* R *be a relation defined as a set of* m *attributes* $(a_1, a_2, ..., a_\mathsf{m})$ *—with* a_m *being the primary key of* R*—, and given* r *an arbitrary set of attributes, the binary mapping of* r *according to* R*, denoted by* $bm_R(\mathsf{r})$*, is defined as follows:*

$$
\begin{aligned}
bm_\mathsf{R}(\mathsf{r}) &= \{b_i\}, 1 \leq i \leq \mathsf{m} \\
b_i &= \begin{cases} 1, & \textit{if } a_i \in \mathsf{r} \\ 0, & \textit{otherwise} \end{cases}
\end{aligned} \tag{2.2}
$$

Using the mapping function above, the vector representation of each one of the query elements in Eq.2.1 (designated henceforth as *segments*), for a dimensional schema comprising one fact table and N dimension tables, is defined as follows:

$$
\begin{aligned}
\mathbf{aggr_q} &= [aggOpCode, bm_\mathsf{Fact}(aggr_q)] \\
\mathbf{proj_q} &= [bm_\mathsf{Fact}(proj_q), bm_\mathsf{Dim1}(proj_q), bm_\mathsf{Dim2}(proj_q), ..., bm_\mathsf{DimN}(proj_q)] \\
\mathbf{join_q} &= [bm_\mathsf{Fact}(join_q), bm_\mathsf{Dim1}(join_q), bm_\mathsf{Dim2}(join_q), ..., bm_\mathsf{DimN}(join_q)] \\
\mathbf{rnge_q} &= [bm_\mathsf{Fact}(rnge_q), bm_\mathsf{Dim1}(rnge_q), bm_\mathsf{Dim2}(rnge_q), ..., bm_\mathsf{DimN}(rnge_q)]
\end{aligned}
$$

where *aggOpCode* designates the aggregate operation using one-hot encoding, namely, `COUNT`: 00001, `SUM`: 00010, `AVG`: 00100, `MAX`: 01000, `MIN`: 10000.

A complete feature vector $\mathbf{q}$ representing a query $q \in \mathcal{Q}$ is set by putting together the above-mentioned segments, that is:

$$\mathbf{q} = [\mathbf{aggr_q}, \mathbf{proj_q}, \mathbf{join_q}, \mathbf{rnge_q}]$$

Accordingly, considering the `SELECT` statement in the example —issued against the SSB (see Fig. 2.2)— a complete feature vector instance is shown below (with $\mathsf{Dim_1}$=Customer, $\mathsf{Dim_2}$=DWDate, $\mathsf{Dim_3}$=Part, and $\mathsf{Dim_4}$= Supplier):

$$\mathbf{q} = [\,[10, 1000000000000]\,, [0, 0, 10000, 1000, 0]\,, \\ [101000, 0, 1, 1, 0]\,, [0, 0, 10000, 0, 0]\,]$$

2.4.2 Query dissimilarity estimation

The collection of feature vectors representing the queries from $\mathcal{Q}$ are arranged as a *representative attribute matrix*, configuring a clustering context $\mathcal{C}$. To be able to identify groupings of related queries in such context, a measure of dissimilarity between observations (vectors) and sets of observations is required. In this sense, a distance function is defined in which similarity between two queries is determined to be proportional to the number of attributes they share in a per-segment (aggregation, projection, join and range predicates) and per-relation (fact and dimensions) basis.

Definition 2.3 ***Segment Distance.*** *Let* $\mathbf{x_p}$ *and* $\mathbf{x_q}$ *be two segments of length* N *belonging to two distinct query vectors* $\mathbf{p}$ *and* $\mathbf{q}$ *from the clustering context* $\mathcal{C}$*. Distance between* $\mathbf{x_p}$ *and* $\mathbf{x_q}$ *(denoted as* $sgmtDst(\mathbf{x_p}, \mathbf{x_q})$*) is defined as:*

$$sgmtDst(\mathbf{x_p}, \mathbf{x_q}) = \frac{1}{N'} \sum_{i}^{N} J(\mathbf{x_{p_i}}, \mathbf{x_{q_i}}), \\ (\mathbf{x_{p_i}}, \mathbf{x_{q_i}}) \neq (0, 0) \tag{2.3}$$

where,

- $J(\mathbf{x_{p_i}}, \mathbf{x_{q_i}})$ is the *Jaccard-Needham* distance estimated between the i-th elements of $\mathbf{x_p}$ and $\mathbf{x_q}$,
- N' is the number of pairs from $\mathbf{x_p}$ and $\mathbf{x_q}$ such that $(\mathbf{x_{p_i}}, \mathbf{x_{q_i}}) \neq (0, 0)$

This way, segment distance is defined as the average *Jaccard-Needham* dissimilarity computed between pairs of segment elements corresponding to those dimensions with at least one attribute being queried (i.e. $\mathbf{x_{p_i}} \neq 0 \vee \mathbf{x_{q_i}} \neq 0$).

The *Jaccard-Needham* distance is a widely-used method for estimating dissimilarity between sets and binary sequences. Unlike similar measurements such as the *Simple Matching coefficient* and the *Rogers-Tanimoto distance*, *Jaccard-Needham* does not count negative co-ocurrences (or mutual absences), which means that null attribute pairs $(0, 0)$ are not rated as matches in the distance estimation. This, in addition to the symmetry of the measure ($J(a, b) = J(b, a)$) and its straightforward computation, makes the *Jaccard-Needham* distance a suitable method for estimating the dissimilarity between the query segments.

Definition 2.4 ***Query Distance.*** *Let* $\mathbf{p}$ *and* $\mathbf{q}$ *be two vectors representing queries from the clustering context* $\mathcal{C}$*. Distance between* $\mathbf{p}$ *and* $\mathbf{q}$ *(denoted as* $qDst(\mathbf{p}, \mathbf{q})$*) is defined as:*

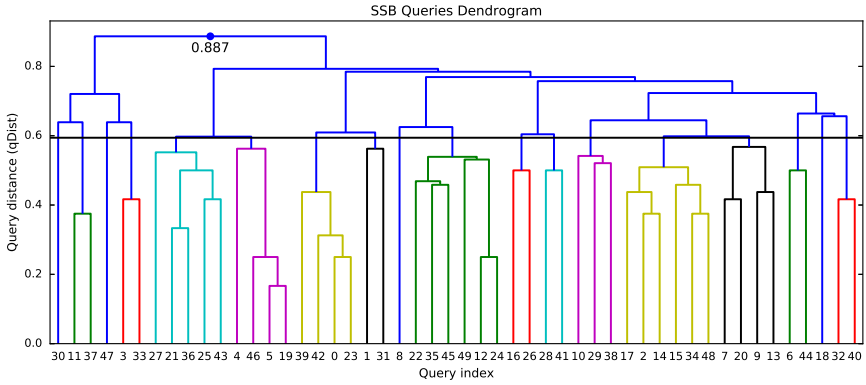
$$\begin{aligned} qDst(\mathbf{p}, \mathbf{q}) = w_{aggr} * sgmtDst(\mathbf{aggr_p}, \mathbf{aggr_q}) \\ + w_{proj} * sgmtDst(\mathbf{proj_p}, \mathbf{proj_q}) \\ + w_{join} * sgmtDst(\mathbf{join_p}, \mathbf{join_q}) \\ + w_{rnge} * sgmtDst(\mathbf{rnge_p}, \mathbf{rnge_q}) \end{aligned} \tag{2.4}$$

where $w_{aggr}, w_{proj}, w_{join}$ and w_{rnge} are arbitrary weights that add up to one (1.0), and condition the influence of each vector segment on the overall dissimiliarity measurement. These weights were estimated when tuning the clustering method the proposed view selection mechanism relies on, by binding their values to the performance of the obtained configuration of clusters estimated in terms of the *F-score* and the *Fowlkes-Mallows index* (*FMI*).

2.4.3 Query clustering and View materialization

The view selection approach documented herein relies on *hierarchical clustering* [37] for deriving groupings of similar queries. In contrast to other well-known clustering methods such as *K-Means* or *K-medoids*, hierarchical clustering analysis does not require to provide the number of clusters upfront. Instead, it generates a hierarchical representation of the entire clustering context in which observations and groups of observations are stacked together from lower to higher levels, according to a distance measure based on the pairwise dissimilarities among the observations. This way, the individual observations lie at the lowest level of the hierarchy as singleton clusters, while at the top level there is only one cluster holding all the observations. As an illustrative example, consider the dendrogram in Fig. 2.3

Figure 2.3 Dendrogram resulting from applying hierarchical clustering analysis on a 50-query workload. Each different color indicates a group of similar queries.



representing the clustering configuration obtained by applying hierarchical clustering on a workload comprising 50 queries.

There are two basic ways to perform hierarchical clustering: through an *agglomerative* procedure (i.e. starting at the bottom and recursively merging pairs of similar clusters into a single cluster, while moving up the hierarchy), or by a *divisive* procedure (i.e. starting at the top of the hierarchy with all observations in one cluster, and recursively partitioning it while moving down the hierarchy). Agglomerative clustering is far more extensively used than its divisive counterpart, hence most of the hierarchical clustering algorithms available fall into this category of methods. In divisive procedures, all the possible partitions of the clustering context are considered in the first step. Since the number of combinations for a collection of N observations is $2^{N-1} - 1$, it is impractical to exhaustively implement these methods and heuristic approximations are used instead [38]. This is why agglomerative clustering was favored over divisive clustering, for analyzing the vectors in the representative attribute matrix of $\mathcal{Q}$.

In order to apply hierarchical agglomerative clustering analysis on a clustering context $\mathcal{C}$ (representative attribute matrix of $\mathcal{Q}$), it is required to specify a *dissimilarity metric* for measuring the distance between pairs of query vectors, and a *linkage criterion* which estimates the dissimilarity among groups of queries as a function of the pairwise distance computed between queries belonging to those groups. Selection of the specific technique to use as linkage criterion was made on the basis of the characteristics of the proposed feature vector representation, and results of preliminary parameter tuning tests (mentioned at the end of section 2.4.2). In consequence—given

that the query vectors derived from $\mathcal{Q}$ do not lie on the Euclidean space—methods such as *centroid*, *median*, and *Ward's linkage* [39] were ruled out. Then, *single linkage*, *complete linkage* and *Weighted Pair Group Method with Arithmetic Mean* (WPGMA) were considered, resulting in the selection of WPGMA, since the former two methods tended to underestimate (single linkage) or overestimate (complete linkage) the dissimilarity between query groupings, according to the mentioned tests. In this way, along with WPGMA, the function defined in the previous section, $qDst$, serves as dissimilarity metric in this case. Under this set-up, the clustering procedure (detailed in algorithm 2.1) starts by assigning each query to its own cluster. Then, the pairwise dissimilarity matrix between such singleton clusters, $\mathbf{D}$, is computed and an empty matrix ($\mathbf{L}$) specifying the resulting dendrogram is initialized. From $\mathbf{D}$, the two most similar (nearest) clusters are merged into one, and appended to $\mathbf{L}$ along with the distance between them (see line 6 in algorithm 2.1). Then, the pairwise dissimilarity matrix gets updated using the WPGMA method for computing the distance between the newly formed cluster and the rest of the currently existing clusters (eq. 2.5):

$$\mathbf{D}[(\mathbf{a} \cup \mathbf{b}), \mathbf{x}] = \frac{qDst(\mathbf{a}, \mathbf{x}) + qDst(\mathbf{b}, \mathbf{x})}{2}, \quad (\mathbf{a}, \mathbf{b} \quad \textit{and} \quad \mathbf{x} \quad \textit{being} \quad \textit{clusters}) \tag{2.5}$$

This procedure is then repeated until there is only one cluster left. Finally, both the clustering configuration ($\mathcal{K}$) and the dendrogram matrix ($\mathbf{L}$) are returned.

Algorithm 2.1 WPGMA clustering procedure

1: $\mathcal{K} \leftarrow \mathcal{C}; \mathcal{C} = \{\mathbf{q_0}, \mathbf{q_1}, \ldots, \mathbf{q_N}\}$ ▷ Initializing clusters (singleton clusters)
2: $\mathbf{D} \leftarrow qDst(\mathbf{q}_i, \mathbf{q}_j)$ for all $\mathbf{q}_i, \mathbf{q}_j \in \mathcal{K}, i \neq j$ ▷ Pairwise dissimilarity matrix
3: $\mathbf{L} \leftarrow []$ ▷ Output matrix
4: **while** $|\mathcal{K}| > 1$ **do**
5: $(\mathbf{a}, \mathbf{b}) \leftarrow argmin(\mathbf{D})$ ▷ Get the nearest clusters
6: append $[\mathbf{a}, \mathbf{b}, \mathbf{D}[\mathbf{a}, \mathbf{b}]]$ to $\mathbf{L}$
7: remove $\mathbf{a}$ and $\mathbf{b}$ from $\mathcal{K}$
8: create new cluster $\mathbf{k} \leftarrow \mathbf{a} \cup \mathbf{b}$ ▷ Merge $\mathbf{a}$ and $\mathbf{b}$ into one cluster
9: update $\mathbf{D}$: $\mathbf{D}[\mathbf{k}, \mathbf{x}] = \mathbf{D}[\mathbf{x}, \mathbf{k}] = \frac{qDst(\mathbf{a}, \mathbf{x}) + qDst(\mathbf{b}, \mathbf{x})}{2}$ for all $\mathbf{x} \in \mathcal{K}$
10: $\mathcal{K} \leftarrow \mathcal{K} \cup \mathbf{k}$
11: **end while**
12: **return** $\mathcal{K}, \mathbf{L}$ ▷ $\mathbf{L}$: WPGMA dendrogram: $((N-1) \times 3)$-matrix

The rationale behind building this clustering configuration is to deduce view definitions containing the queries grouped under each cluster, so that for each cluster $\mathbf{c}_i \in \mathcal{K}$ there exists a view $V_i \in \mathcal{V}$ able to answer the queries in $\mathbf{c}_i$. However, as stated earlier, to avoid further processing of clusters

grouping queries that are not closely related (i.e. spurious clusters), a score was defined indicating to what extent it is worth to materialize the view derived from a particular cluster.

Definition 2.5 Materializable cluster. *A cluster* $\mathbf{c}$ *from a clustering configuration* $\mathcal{K}$ *is said to be* materializable *if the following conditions are met:*

1. *Queries in* $\mathbf{c}$ *are highly similar to each other.*
2. *Queries in* $\mathbf{c}$ *are clearly separated (highly dissimilar) from queries in other clusters.*
3. $\mathbf{c}$ *covers as many queries as possible. In other words,* $|\mathbf{c}|$ *is large enough in proportion to the size of the workload (*$|\mathcal{Q}|$*)*

For a cluster to meet the first two conditions it should be *consistent*, while the third condition prevents singleton and small clusters from being further processed. Based on the above definition, the *materializable score* of a cluster ($mat(\mathbf{c})$ in eq. 2.6) is computed as the product of two sigmoid functions: one on the per-cluster *silhouette score* (S) [40]—defined below in eq. 2.7—and the other on the per-cluster proportions (P).

$$mat(\mathbf{c}) = \Big(\frac{1}{1+e^{-k(S(\mathbf{c})-s_0)}}\Big)\Big(\frac{1}{1+e^{-k(P(\mathbf{c})-p_0)}}\Big) \tag{2.6}$$

with:

$$S(\mathbf{c}) = \frac{1}{|\mathbf{c}|}\sum_{\mathbf{q}_i\in\mathbf{c}}\frac{b(\mathbf{q}_i)-a(\mathbf{q}_i)}{max\left\{a(\mathbf{q}_i),b(\mathbf{q}_i)\right\}} \quad , \quad P(\mathbf{c}) = \frac{|\mathbf{c}|}{|\mathcal{Q}|} \tag{2.7}$$

where,

- k is a factor that controls the steepness of both of the sigmoid functions,
- s_0 and p_0 are the midpoints of the silhouette and cluster-proportion sigmoids respectively,
- $a(\mathbf{q}_i)$ is the average distance between $\mathbf{q}_i$ and all queries within the same cluster,
- $b(\mathbf{q}_i)$ is the lowest average distance of $\mathbf{q}_i$ to all queries in any other clusters.

By setting a fixed threshold on this score ($S \geq 0.5$ by default) it is possible to unambiguously set the spurious clusters apart from those whose corresponding views are worth materializing. The next step is deriving view definitions covering the queries arranged under each of the *materializable clusters* ($\mathcal{K}_{mat} \subseteq \mathcal{K}$), this is to say:

$$\forall \mathbf{c}_i \in \mathcal{K}_{mat}, \exists V_i | \forall \mathbf{q} \in \mathbf{c}_i, \mathbf{q} \subseteq V_i$$

Algorithm 2.2 below details the procedure conducted to derive the views V_i meeting this containment condition on each of the materializable clusters. In this procedure, the SPJ-query clauses (*aggregate*, *projection*, *join predicates*, and *group-by*) of the resulting views are defined in terms of the union of the corresponding attributes from each query in the cluster.

Algorithm 2.2 Procedure for deriving view definitions

Let $\mathbf{c}$ be a cluster in $\mathcal{K}_{mat}$
$V \leftarrow [aggr_V, proj_V, join_V, groupBy_V]$ ▷ Output view definition
for each query $\mathbf{q}$ in $\mathbf{c}$ **do**
    $aggr_V \leftarrow aggr_V \cup aggr_{\mathbf{q}}$
    $proj_V \leftarrow proj_V \cup proj_{\mathbf{q}} \cup rnge_{\mathbf{q}}$
    $join_V \leftarrow join_V \cup join_{\mathbf{q}}$
    $groupBy_V \leftarrow groupBy_V \cup proj_{\mathbf{q}} \cup rnge_{\mathbf{q}}$
end for
return V

It is worth noting that view definitions are generated without *range predicates*. Instead, the attributes used in this query clause are pushed to the *projection* and *group-by* clauses of the view definition. In this way, the resulting materialized views are able to answer not only the queries grouped under each cluster, but also unseen queries with arbitrary range predicates on the attributes listed in the projection clause of the view definition[1]. Additionally, thanks to the iterative nature of the underlying data transformation framework, further optimization of the derived views is possible. Later in this chapter (Section 2.6.3) a view maintenance strategy is discussed which leverages on the continuous data transformation process described back in section 2.3.

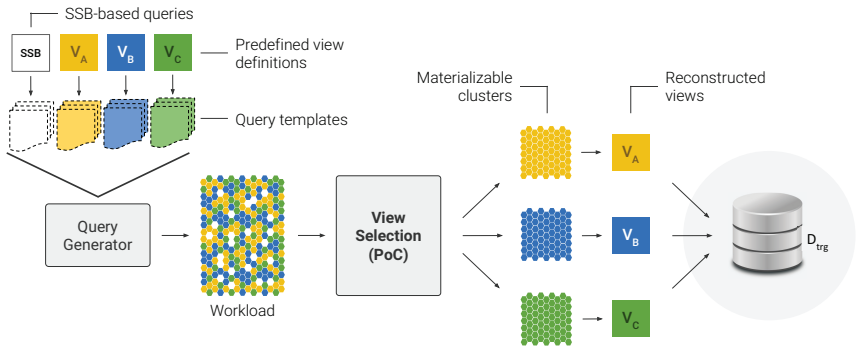
To recap, this section developed a thorough description of the syntactic analysis of query workloads that makes up the view selection mechanism proposed here. It started by specifying the procedure for obtaining a structured representation of the queries in the form of a feature vector and a representative attribute matrix. Then, a custom query dissimilarity function was defined, tailored specifically to the structure and values of said feature vectors. Finally, a query clustering algorithm based on the pairwise

[1] Although this only applies for queries with distributive aggregate functions, i.e. `SUM`, `COUNT`, `MIN`, and `MAX`.

dissimilarities between the analyzed queries was addressed, detailing as well the procedure for deriving materialized view definitions out of the obtained clustering configuration.

2.5 Proof-of-concept Implementation: Star Schema Benchmark (SSB) and workload generation

Figure 2.4 Proof-of-concept implementation of the proposed view selection mechanism.



A bottom-up approach was adopted to test the principles, assumptions and procedures governing the view selection mechanism detailed in the previous sections. In this way, starting from a set of predefined view definitions, the effectiveness of the proposed mechanism is estimated in terms of its ability for identifying the same set of views and reconstructing their definitions, upon analyzing a query workload generated from query templates fitting the original set of views (see Figure 2.4).

As stated back in section 2.4, this research leverages on the *Star Schema Benchmark* (SSB) as baseline schema and dataset, and therefore both the predefined views and query templates, as well as the query generator module were designed and built so they conform to the data model the SSB embodies.

The SSB comprises a dimensional data model (four dimensions, one fact table), an extensible dataset (size depending on a *scaling factor*—*SF*), and a set of queries typical for data warehouse applications arranged in four categories/families designated as *Query Flights* (a detailed definition of the SSB is available online [36]).

Thirteen Select-Project-Join query statements in total compose the full

query set of the SSB. For the proof-of-concept that is being described, three view definitions were derived based on the original SSB query set, and from each view definition, four query templates were prepared. Additionally, one template for each one of the 13 canonical SSB queries was also composed. With this set of 25 templates as input, a module that generates random instances of runnable queries enabled the creation of query workloads of arbitrary size. Listings below present the definitions of each one of the mentioned views.

```
SELECT sum(lo_revenue), p_brand1, c_region,
       s_region, d_year
FROM lineorder, customer, dwdate, part, supplier
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY p_brand1, c_region, s_region, d_year
ORDER BY p_brand1, c_region, s_region, d_year
```

Listing 2.2: Definition of View A

```
SELECT sum(lo_ordtotalprice), p_category, c_city,
       s_city, d_yearmonthnum
FROM lineorder, customer, dwdate, part, supplier
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY p_category, c_city, s_city, d_yearmonthnum
ORDER BY p_category, c_city, s_city, d_yearmonthnum
```

Listing 2.3: Definition of View B

```
SELECT sum(lo_supplycost - lo_tax), c_region, p_mfgr,
       s_region, c_nation, d_year
FROM lineorder, customer, dwdate, part, supplier
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY c_region, p_mfgr, s_region, c_nation, d_year
ORDER BY c_region, p_mfgr, s_region, c_nation, d_year
```

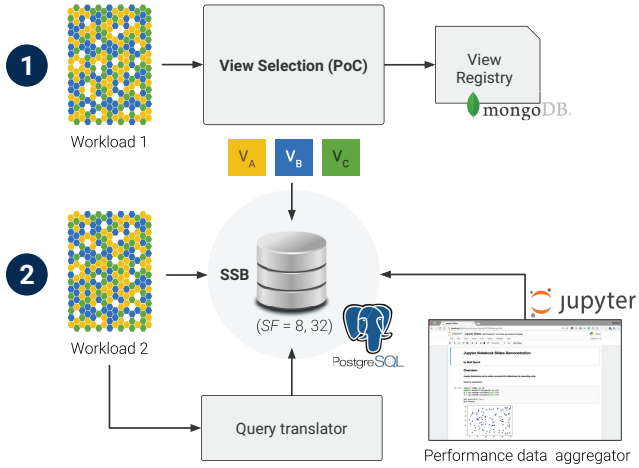
Listing 2.4: Definition of View C

2.6 Experimental Evaluation

2.6.1 Experimental setup

Figure 2.5 depicts the arrangement of components and technologies used for conducting the experimental evaluation of the proposed view selection approach. This evaluation comprised three main stages:

Figure 2.5 Experiment set-up



(*1*) Running the view selection implementation on a 400-query workload to the point that it materializes views A, B, and C (defined earlier in section 2.5), while keeping track of the runtime involved in the procedures of *query clustering*, *view scoring* (using the materializable score defined in section 2.4.3), *view definition*, *view registering* and *view creation (i.e. materialization)*.

(*2*) Once the views are materialized, run a second 400-query workload against both the base SSB dataset and the materialized views. In doing the latter, queries first pass through a translation component that gathers the details of the available materialized views from the *view registry* (stored in a *MongoBD*[2] document database), and adapts the incoming query statements accordingly.

(*3*) Running the two previous stages on workloads of different sizes and query distributions.

For all the stages, the performance information collected from running the tests were aggregated and visualized using *Jupyter notebook*[3]. During this evaluation, workloads of multiple sizes were run against two different dimensions of the SSB dataset: 48 million rows ($SF = 8$), and 192 million rows ($SF = 32$). These datasets were stored into *PostgreSQL*[4] databases

[2]Available at mongodb.com
[3]Available at jupyter.org

deployed on two VMWare® virtual machines with the following specifications: Intel® Xeon® E5645 @2.40GHz CPU, 20GB RAM, 500GB hard disk.

2.6.2 Results

2.6.2.1 View selection overhead

Running the view selection implementation on a validation workload (400 queries) and a SSB dataset with $SF = 32$ took around 4200 seconds in total (i.e. 1 hour and 10 minutes). While this might be deemed as a considerable amount of time, it is worth mentioning that the actual analysis of the workload takes just a small fraction of it. As mentioned before, the view selection process involves the execution of a sequence of steps: (1) *query clustering*, (2) *view (or cluster) scoring*, (3) *view definition*, (4) *view registering* and (5) *view creation*. Out of these only the first four steps have to do with the syntactical analysis of query sets described throughout this chapter, while the last one (view creation) refers to the actual materialization of the derived views in the data store.

Figure 2.6 Materialized view selection runtime per stage (SSB $SF = 32$, $|\mathcal{Q}| = 400$)

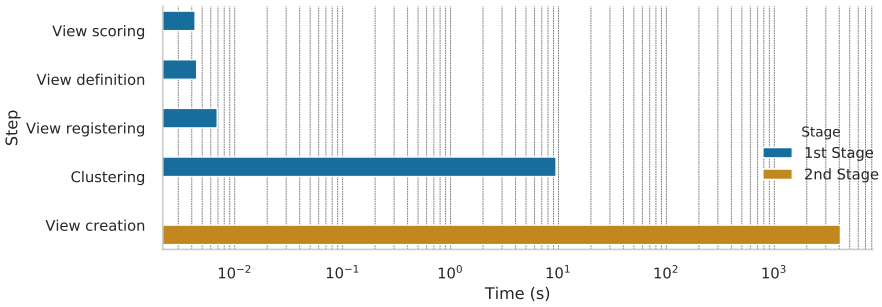


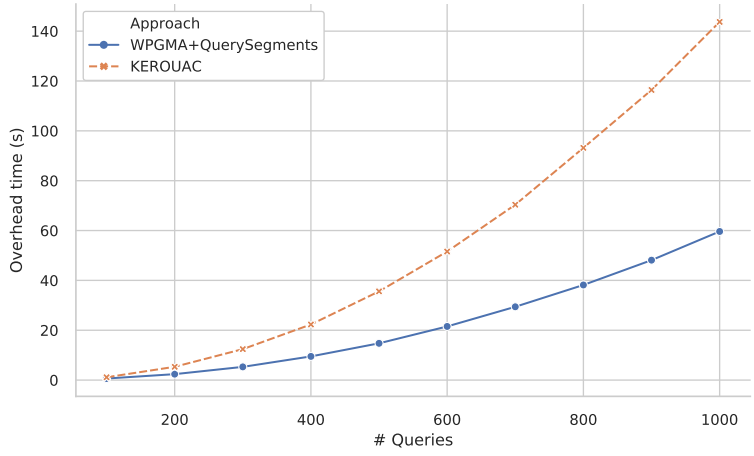
Figure 2.6 portrays the substantial difference between these execution times by defining two stages: the first one aggregates the initial four steps, and the second one comprises the view creation step only. Note how the steps from stage one amounts to less than 10 seconds, adding up to just the 0.23% of the total execution time, while the remaining 99.77% is the time it takes for the data store (PostgresSQL in this case) to build and persist the resulting views.

While in classical data-mining methods for view selection, query sets are mapped into a *query vs attribute* matrix (where statements are represented

[4]PostgreSQL v9.5.8 working with the default configuration (postgresql.org)

as flat binary sequences), the method proposed in this work accounts not only for query-attribute usage, but also for the basic structure of analytical queries (see *query segments* in section 2.4.1). The *KEROUAC* method by Aouiche et al.[21, 29] discussed back in section 2.2, adopts this classical approach and additionally relies on a clustering technique that can be regarded as *divisive clustering*, in the sense that it proceeds by first placing the whole collection of queries into a single cluster, and then iteratively partitioning it until convergence to a stable clustering configuration is achieved. Given that the mechanism proposed herein adopts a *agglomerative clustering* technique, it was considered relevant to contrast the performance of both methods. To this end, authors of KEROUAC were contacted to provide missing information required to replicate their approach. As a result, Figure 2.7 shows the variation of the overhead time of the proposed method w.r.t. the workload size, and compares it to that from KEROUAC. While the approach proposed herein features a quadratic rate of growth—since the implementation of the WPGMA method used in the clustering analysis relies on the *nearest-neighbors chain algorithm* which is known to run in $O(N^2)$ time [39]—it still outperforms similar methods running under the same conditions, as evidenced in Figure 2.7.

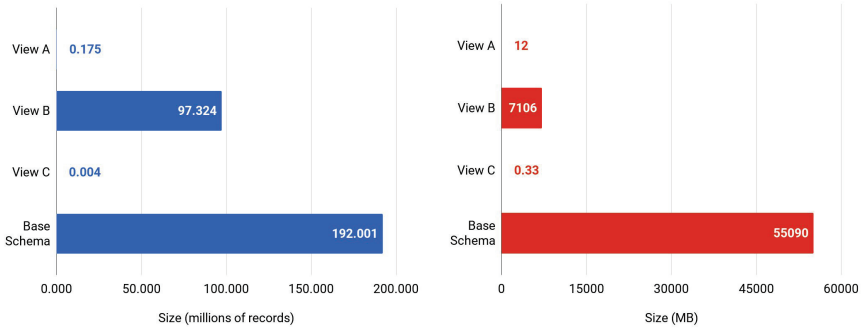
Figure 2.7 View selection overhead vs Workload size.



When it comes to storage cost, view size varies depending on the cardinality of the fields used in the group-by clause of their definition [21], and whether or not there are hierarchical relations between such attributes (e.g. the one between `c_region`, `c_nation` and `c_city`). Figure 2.8a shows the amount of records per materialized view, in contrast to the number of rows in the SSB base schema. While the amount of records of views

A (175.000) and C (4.375) is fairly negligible in comparison with the base schema (192.000.754 records), view B amounts to almost half the size of the base dataset. Nonetheless, in terms of disk space usage the proportion between views and base schema is more favorable, as evidenced in Figure 2.8b. This way, while the amount of records stored into the three derived views add up to 50% of the number of rows in the base schema, the disk space used by said views is only 13% the size of the base data collection.

Figure 2.8 Materialized views size: (a) millions of records. (b) disk space. — ($SSB\ SF = 32$).



2.6.2.2 View selection performance

With the selected views already materialized, a 400-query workload was run against the base SSB dataset ($SF = 32$) to get a query latency baseline. Out of those 400 queries, 300 were covered by the three available materialized views (100 queries per view), and the remaining ones were canonical SSB-based queries. Once the latency baseline was built, the same workload was issued this time with the query translation module in place, so that incoming queries matching any of the definitions of the available materialized views get rewritten and issued against them. Figure 2.9 illustrates the contrast between the baseline query latency (`baseline_time`) and the latency when queries run against the materialized views (`runtime`). For queries that benefit from views A and C the gain in query latency is remarkably substantial, going from hundreds of seconds in the baseline to sub-second runtime in the views. On the other hand, queries running on the view B present an important—though less striking—reduction in latency. In this

case, the size of this view, which in terms of number of records is comparable to the base dataset, prevents queries from running under interactive latency constraints.

Figure 2.9 Query runtime per view: Baseline runtime vs. View runtime ($SSB\ SF = 32$, $|\mathcal{Q}| = 400$)

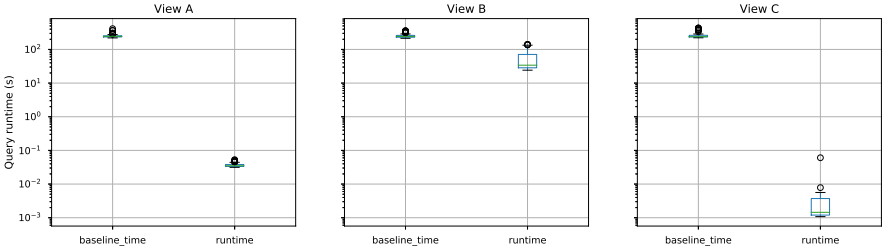


Table 2.1 summarizes the results obtained from running the above test, including the reduction in latency achieved through each one of the views, relative to the average baseline query runtime.

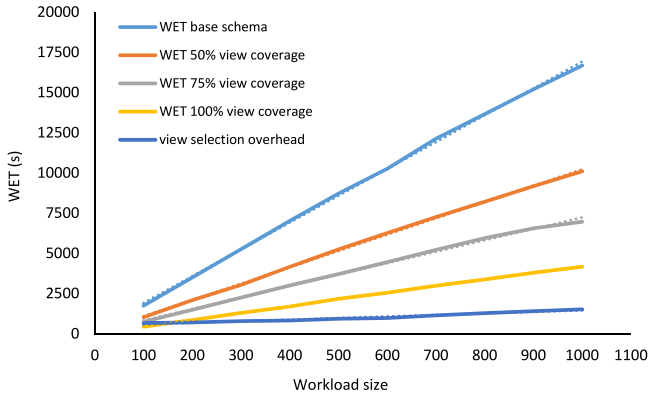
Table 2.1: Query latency reduction per view ($SSB\ SF = 32$, $|\mathcal{Q}| = 400$)

	Baseline time (s)	Translation time (s)	View runtime (s)	% Latency reduction
View A	251.04±2.85	$(3.54\pm0.11)\times10^{-3}$	$(36.89\pm0.49)\times10^{-3}$	99.98
View B	253.53±3.53	$(10.7\pm0.45)\times10^{-3}$	51.83±3.22	79.55
View C	256.98±4.51	$(5.18\pm0.45)\times10^{-3}$	$(2.86\pm0.21)\times10^{-3}$	99.99

2.6.2.3 Response to workload size

The purpose of this last evaluation was estimating the influence of the view selection mechanism in the workload execution time (hereafter WET), using query sets of multiple sizes and different query distributions. As with the previous tests, each workload was first run against the base SSB dataset ($SF = 8$), this time measuring the execution time for the full workload instead of the average query runtime. Afterwards, each workload was fed to the view selection implementation, measuring again the WET, as well as the view selection overhead. Three query distributions were considered for the workloads in this evaluation: (*1*) half the queries in the workload are covered by the materialized views, (*2*) 75% of the queries in the workload are covered by the materialized views, and (*3*) all the queries in the workload are covered by the available materialized views. Figure 2.10 presents the results obtained for different workload sizes and query distributions.
Results displayed in Figure 2.10 evidence a consistent decrease in the WET as the amount of queries covered by the materialized views grows: for the

Figure 2.10 Workload execution time (WET) vs. Workload size (SSB $SF = 8$)



50%-view-covered workloads the WET was consistently reduced by about 40%, while the 75%-view-covered and the full-view-covered workloads took 57% and 75% less time to run respectively. As the results from the previous tests indicated, the drop in the WET is closely tied to the size of the materialized views. Under the configuration used for this test, the aggregated size of the materialized views add up to 85% of the number of records of the base dataset and 46% of the disk space this last one uses. This way, in order to further reduce the workload execution time in cases like this, a better compromise between view's query coverage and view size has to be achieved. The next section elaborates on this constraint and outlines a practical strategy to address it, leveraging on the iterative nature of the dynamic transformation framework upon which the proposed view selection mechanism was conceived.

2.6.3 Discussion

The syntactic analysis lying at the core of the mechanism described in this chapter, proved an effective method for identifying groups of related queries and deriving a limited but comprehensive set of materialized views out of them. The experimental evaluation conducted on a proof-of-concept implementation of the devised approach reported that, despite the appreciable overhead, the time it takes for the procedure to run is nearly linear on the number of queries in the workload. Moreover, most of the overhead in time is due to the materialization of the selected views in the underlying storage technology, while the time spent in the actual analysis of the workload is relatively negligible.

On the other hand, when examining the effect of using the resulting materialized views on the query latency and workload execution time, there is an evident and substantial improvement considering a drop in query latency ranging from 80% to 99.99%, and a decrease in the WET of 40% to 75% depending on the view coverage of the workload. However, there are cases where maximizing the query coverage of the derived views might lead to a prohibitively large storage overhead critically impacting the relative benefit of using materialized views. By leveraging on the iterativity inherent to the dynamic data transformation framework introduced at the beginning of section 2.3, it is possible to cope with the stated limitation, considering the following extra steps:

First iteration, before materializing a selected view:

1. Estimate the maximum size of the view as the product of the cardinalities of the attributes listed under the *group-by* clause of its definition.

2. When the estimated size is comparable to the amount of records of the fact table, partition the view on the attribute with the less cardinality, this way ending up with a set of size-bounded child views.

Subsequent iterations:

1. In the query translation component, anytime an incoming query matches a partitioned view rewrite it into several sub-queries targeting each one a different partition. Then, run such sub-queries concurrently and aggregate their result sets.

2. Identify *hot regions* in the available materialized views (i.e. sets of tuples that get queried the most) leveraging on the continuous monitoring of the incoming queries.

3. Use horizontal partitioning on the views to set apart the *hot regions* from the remaining tuples.

4. Keep track of the *hit ratio*[5] of each materialized view and its corresponding partitions to eventually dispose of those views/partitions that seldom get queried.

The above procedures configure a practical maintenance strategy that allows to address the storage overhead constraints and maximize the benefit of the

[5]Ratio of the number of queries answered by the view/partition to the total number of queries in a certain period of time

materialized views the view selection mechanism comes up with. This strategy is currently under development and its integration to the mechanism proposed in this chapter has been deferred to future work.

2.7 Conclusions and Future Work

Organizations nowadays face a daunting challenge when trying to make sense of the massive and ever-growing amount of data generated in their day-to-day operation. Being able to conduct ad-hoc querying and get visual insights from such data in an efficient and timely manner is key for business to support their decisions. In this regard, this chapter describes an approach for automatically generating materialized views to speed up data retrieval on dimensionally modeled datasets. The proposed approach has been conceived as part of a framework for dynamic data transformation intended to generate read-optimized data schemas, and relies on syntactic analysis of query workloads issued against the data collection. The developed method provides a way to estimate how similar two queries are, and put together clusters of related queries based on said estimation. Then, the method is able to tell out of those clusters which ones are worth materializing—based on consistency and query coverage criteria—and derive a view definition for each *materializable* cluster, based on the queries they group.

The experimental evaluation conducted on a proof-of-concept implementation was focused on measuring the overhead the proposed view selection method entails and contrasting it to the relative benefit it brings in return. Results show that in general such processing overhead pays off in query latency, leading to a drop ranging from 80% to 99.99% at the expense of 13% of the disk space used by the base dataset for persisting the derived materialized views. The evaluation also reported a few open challenges of the proposed approach when it comes to finding an adequate level of query coverage, so that the storage overhead due to the views does not compromise the benefit of using them. To deal with the declared limitation a preliminary view maintenance strategy was outlined, resorting to the iterative optimization of the available views by using horizontal partitioning and keeping track of their usage patterns over time.

In consequence, upcoming work on this research will extend the current view selection mechanism with the view maintenance strategy described earlier and evaluate its impact on query performance. Additionally, the dynamic data transformation framework—that lays the foundations for the approach proposed in this chapter—will be further developed to relax the

assumption of temporary immutability of the base dataset, and also incorporate polyglot persistence (i.e. intelligently scatter data over multiple data store technologies).

Acknowledgements

This work was supported by the Research Foundation Flanders (FWO) under Grant number G059615N - "*Service oriented management of a virtualised future internet*".

Addendum

***Note on the storage cost*:** It is well known that view materialization entails a processing and storage overhead. One of the assumptions made by the mechanism proposed in this chapter to identify and materialize candidate views states that *latency improvement is favored over storage costs* (cf. section 2.3.2). Said assumption should not be taken as meaning that the storage overhead is not a concern for the view selection mechanism, otherwise the system might as well create one view for each of the statements in the workload. Quite conversely, by discovering groups of similar queries and deriving representative view definitions out of said groups, the method presented in this chapter aims at identifying a limited set of views that can serve as many queries from the workload as possible. In that sense, while the proposed view selection mechanism does not intend to achieve the optimal trade-off between query processing cost and storage overhead, it does attempt to reduce the amount of views materialized for a given workload.

***Note on scalability*:** Concerning the related approaches, one of the main issues identified in this chapter is their lack of scalability (section 2.2). The experimental evaluation conducted on the view selection mechanism shows that the proposed method responds better than similar solutions to the increase in the workload size (Figure 2.7). However, these results might be deemed as providing an incomplete picture of the scalability of the proposed approach, especially considering that tests were made on a setup with two instaces of conventional PostgreSQL servers. In this sense, the work presented in Appendix A of this dissertation elaborates on the applicability of the devised view selection mechanism to a scenario with distributed data storage, using an infrastructure based on the Apache Hadoop framework with

Apache Spark for data ingestion and query processing. Moreover, the evaluation conducted in this environment allowed estimating the perceived benefits and costs of the proposed approach as the volume of records in the base dataset increases. While the obtained results show that devised mechanism can substatially reduce the query processing time for distributed data collections, challenges concerning the maintenance and incremental update of the derived materialized views are still to be addressed.

***Note on database performance tunning and indexes*:** The original manuscript this chapter is based upon does not report some relevant details concerning the configuration of the database and the indexes used in the experimental evaluation. On the one hand, no performance tuning was conducted on the data stores, thus default configuration parameters were used for both instances of PostgreSQL. On the other hand, only conventional B-tree indexes were created on each of the primary keys of the entities composing the SSB data schema. In this regard, it would be certainly possible to further improve the performance of analytical queries running against the base data collection by using data structures such as Bitmap indexes on low-cardinality attributes [41].

References

[1] Nathan Marz and James Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., USA, 1st edition, 2015. ISBN 1617290343.

[2] Raul Castro Fernandez, Peter R. Pietzuch, Jay Kreps, Neha Narkhede, Jun Rao, Joel Koshy, Dong Lin, Chris Riccomini, and Guozhang Wang. Liquid: Unifying Nearline and Offline Big Data Integration. In *CIDR 2015*.

[3] Benoy Anthony, Konstantin Boudnik, Cheryl Adams, Branky Shao, Cazen Lee, and Kai Sasaki. In-Memory Computing in Hadoop Stack. In *Professional Hadoop®*, pages 161–182. Wiley Online Library, 2016.

[4] Sumit Pal. SQL-on-Big-Data Challenges & Solutions. In *SQL on Big Data: Technology, Architecture, and Innovation*, pages 17–33. Apress, Berkeley, CA, 2016. ISBN 978-1-4842-2247-8. doi: 10.1007/978-1-4842-2247-8_2.

[5] AMPLab-UC-Berkeley. AMPLab Big Data Benchmark, 2014. URL https://amplab.cs.berkeley.edu/benchmark/. https://amplab.cs.berkeley.edu/benchmark/. Last accessed: 2017-04-18.

[6] Leandro Ordonez-Ante, Thomas Vanhove, Gregory Van Seghbroeck, Tim Wauters, and Filip De Turck. Interactive Querying and Data Visualization for Abuse Detection in Social Network Sites. In *ICITST 2016*, pages 104–109. IEEE, 2016.

[7] Leandro Ordonez-Ante, Thomas Vanhove, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert, and Filip De Turck. Dynamic Data Transformation for Low-Latency Querying in Big Data Systems. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2480–2489, Dec 2017. doi: 10.1109/BigData.2017.8258206.

[8] Thomas Vanhove, Merlijn Sebrechts, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert, and Filip De Turck. Data transformation as a means towards dynamic data storage and polyglot persistence. *International Journal of Network Management*, 27(4):e1976, 2017. doi: 10.1002/nem.1976.

[9] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley Publishing, 3rd edition, 2013. ISBN 1118530802, 9781118530801.

[10] Daihee Park, Jaehak Yu, Jun-Sang Park, and Myung-Sup Kim. NetCube: a comprehensive network traffic analysis model based on multidimensional OLAP data cube. *International Journal of Network Management*, 23(2):101–118, 2013.

[11] O. Ali, P. Crvenkovski, and H. Johnson. Using a business intelligence data analytics solution in healthcare. In *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–6, Oct 2016. doi: 10.1109/IEMCON.2016.7746328.

[12] Michael Scriney, Martin F. O'Connor, and Mark Roantree. Generating Cubes from Smart City Web Data. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '17, pages 49:1–49:8, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4768-6. doi: 10.1145/3014812.3014863.

[13] T Nalini, A Kumaravel, and K Rangarajan. A Comparative Study Analysis of Materialized View for Selection Cost. *World Applied Sciences Journal (WASJ)*, 20(4):496–501, 2012.

[14] Rajib Goswami, Dhruba Kr Bhattacharyya, Malayananda Dutta, and Jugal K. Kalita. Approaches and Issues in View Selection for Materialising in Data Warehouse. *International Journal of Business Information Systems*, 21(1):17–47, December 2016. ISSN 1746-0972. doi: 10.1504/IJBIS.2016.073379.

[15] Anjana Gosain and Kavita Sachdeva. A Systematic Review on Materialized View Selection. In Suresh Chandra Satapathy, Vikrant Bhateja, Siba K. Udgata, and Prasant Kumar Pattnaik, editors, *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*, pages 663–671, Singapore, 2017. Springer Singapore. ISBN 978-981-10-3153-3.

[16] Thomas P. Nadeau and Toby J. Teorey. Achieving Scalability in OLAP Materialized View Selection. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '02, pages 28–34, New York, NY, USA, 2002. ACM. ISBN 1-58113-590-4. doi: 10.1145/583890.583895.

[17] Maria Trinidad Serna-Encinas and Jose Antonio Hoyo-Montano. Algorithm for selection of materialized views: based on a costs model. In *International Conference on Current Trends in Computer Science, 2007. ENC 2007.*, pages 18–24. IEEE, 2007.

[18] Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize in a data warehouse. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):24–43, 2005.

[19] Jiratta Phuboon-ob and Raweewan Auepanwiriyakul. Two-Phase Optimization for Selecting Materialized Views in a Data Warehouse . *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 1(1):119–123, 2007. ISSN eISSN:1307-6892. URL http://waset.org/Publications?p=1.

[20] Hossein Azgomi and Mohammad Karim Sohrabi. A game theory based framework for materialized view selection in data warehouses. *Engineering Applications of Artificial Intelligence*, 71: 125–137, 2018. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2018.02.018. URL http://www.sciencedirect.com/science/article/pii/S0952197618300472.

[21] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. Clustering-Based Materialized View Selection in Data Warehouses. In *Advances in Databases and Information Systems*, pages 81–95, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-37900-3.

[22] Roozbeh Derakhshan, Bela Stantic, Othmar Korn, and Frank Dehne. Parallel simulated annealing for materialized view selection in data warehousing environments. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 121–132. Springer, 2008.

[23] Xia Sun and Ziqiang Wang. An efficient materialized views selection algorithm based on PSO. In *Intelligent Systems and Applications, 2009. ISA 2009. International Workshop on*, pages 1–4. IEEE, month 2009.

[24] Qingzhou Zhang, Xia Sun, and Ziqiang Wang. An efficient ma-based materialized views selection algorithm. In *Control, Automation and Systems Engineering, 2009. CASE 2009. IITA International Conference on*, pages 315–318. IEEE, 2009.

[25] Rajib Goswami, DK Bhattacharyya, and Malayananda Dutta. Materialized view selection using evolutionary algorithm for speeding up big data query processing. *Journal of Intelligent Information Systems*, 49 (3):407–433, 2017.

[26] Anjana Gosain and Kavita Sachdeva. Materialized View Selection Using Backtracking Search Optimization Algorithm. In *Intelligent Engineering Informatics*, pages 241–251. Springer, 2018.

[27] T.V. Vijay Kumar and Amit Kumar. Materialized View Selection Using Set Based Particle Swarm Optimization. *Int. J. Cogn. Inform. Nat. Intell.*, 12(3):18–39, July 2018. ISSN 1557-3958. doi: 10.4018/IJCINI.2018070102. URL https://doi.org/10.4018/IJCINI.2018070102.

[28] Jay Prakash and TV Vijay Kumar. A Multi-Objective Approach for Materialized View Selection. *International Journal of Operations Research and Information Systems (IJORIS)*, 10(2):1–19, 2019.

[29] Kamel Aouiche and Jérôme Darmont. Data mining-based materialized view and index selection in data warehouses. *Journal of Intelligent Information Systems*, 33(1):65–93, 2009.

[30] T. V. Vijay Kumar, Archana Singh, and Gaurav Dubey. Mining Queries for Constructing Materialized Views in a Data Warehouse. In David C. Wyld, Jan Zizka, and Dhinaharan Nagamalai, editors, *Advances in Computer Science, Engineering & Applications*, pages 149–159, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-30111-7.

[31] P Jouve and Nicolas Nicoloyannis. KEROUAC: an Algorithm for Clustering Categorical Data Sets with Practical Advantages. In *International Workshop on Data Mining for Actionable Knowledge (DMAK'2003, in conjunction with PAKDD03)*, volume 70, 2003.

[32] Pierre-Emmanuel Jouve. *Apprentissage Non Supervisé et Extraction de Connaissances à partir de Données*. PhD thesis, Université Lumière-Lyon 2, 2003.

[33] Olcay Taner Yıldız and Onur Dikmen. Parallel Univariate Decision Trees. *Pattern Recognition Letters*, 28(7):825–832, 2007.

[34] Jiang Du, Boris Glavic, Wei Tan, and Renée J. Miller. DeepSea: Progressive Workload-Aware Partitioning of Materialized Views in Scalable Data Analytics. In *International Conference on Extending Database Technology 2017*, pages 198–209. OpenProceedings.org, 2017. ISBN 978-3-89318-073-8.

[35] Rada Chirkova, Alon Y Halevy, and Dan Suciu. A formal perspective on the view selection problem. In *27th International Conference on Very Large Data Bases*, volume 1, pages 59–68, 2001.

[36] Patrick E O'Neil, Elizabeth J O'Neil, and Xuedong Chen. The Star Schema Benchmark (SSB), 2009. URL https://www.cs.umb.edu/~poneil/StarSchemaB.PDF.

https://www.cs.umb.edu/~poneil/StarSchemaB.PDF. Last accessed: 2018-11-28.

[37] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Clustering Analysis. In *The elements of statistical learning: Data mining, inference and prediction*, chapter 14, pages 501–520. Springer series in statistics, New York, 2009.

[38] Alok Sharma, Yosvany López, and Tatsuhiko Tsunoda. Divisive hierarchical maximum likelihood clustering. *BMC bioinformatics*, 18(16): 546, 2017.

[39] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *Computing Research Repository (CoRR)*, abs/1109.2378, 2011. URL http://arxiv.org/abs/1109.2378.

[40] Peter J Rousseeuw. Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of computational and applied mathematics*, 20(1):53–65, 1987.

[41] Morteza Zaker, Somnuk Phon-Amnuaisuk, and Su-Cheng Haw. An Adequate Design for Large Data Warehouse Systems: Bitmap index versus B-tree index. *International Journal of Computers and Communications*, 2(2):39–46, 2008.

3

EXPLORA: Interactive Querying of Multidimensional Data in the Context of Smart Cities

The methods presented in the previous chapter assumed the dataset to be temporarily immutable. The approach presented in this chapter relaxes said assumption and introduces EXPLORA*, a data processing framework able to precompute the results of recurrent queries on an ever-growing dataset generated by sensors deployed in a smart city environment. The motivation behind* EXPLORA *consists in enabling low latency querying for interactive data exploration applications on top of live datasets. In that sense, this framework incorporates a processing pipeline that incrementally computes continuous views over the stream of data. These views are optimized to serve queries representing common interaction patterns adopted by users of this kind of applications. Besides elaborating on the architecture and design decisions of the proposed approach, this chapter presents two proof-of-concept implementations of* EXPLORA *operating on data from a real smart city setup. The performance evaluation conducted in these implementations shows a substantial reduction in query response time in contrast to queries running on the collection of raw sensor measurements.*

⋆⋆⋆

L. Ordonez-Ante, G. Van Seghbroeck, T. Wauters, B. Volckaert and F. De Turck

Published as special issue paper in the MDPI Sensors Journal, May 2020.

Abstract Citizen engagement is one of the key factors for smart city initiatives to remain sustainable over time. This in turn entails providing citizens and other relevant stakeholders with the latest data and tools that enable them to derive insights that add value to their day-to-day life. The massive volume of data being constantly produced in these smart city environments makes satisfying this requirement particularly challenging. This chapter introduces EXPLORA, a generic framework for serving interactive low-latency requests, typical of visual exploratory applications on spatio-temporal data, which leverages the stream processing for deriving—on *ingestion time*—synopsis data structures that concisely capture the spatial and temporal trends and dynamics of the sensed variables and serve as compacted data sets to provide fast (approximate) answers to visual queries on smart city data. The experimental evaluation conducted on proof-of-concept implementations of EXPLORA, based on traditional database and distributed data processing setups, accounts for a decrease of up to 2 orders of magnitude in query latency compared to queries running on the base raw data at the expense of less than 10% query accuracy and 30% data footprint. The implementation of the framework on real smart city data along with the obtained experimental results prove the feasibility of the proposed approach.

3.1 Introduction

The increasing pervasiveness of data in the world is currently leading to a new era of human progress, which has been referred to as the *Fourth Industrial Revolution*. As part of this new dynamic, initiatives in the context of smart cities have emerged, aiming at harnessing the power of data to connect with citizens, to build public awareness, to drive urban development and local public policy, and to answer pressing problems such as how to lighten the huge strain that human development has historically placed on the environment and Earth's natural resources. The burgeoning information technology (IT) industry has played a major role in bringing forth these kind of initiatives: big data, Internet of Things (IoT), and cloud computing technologies are at the core of the smart city strategies being implemented nowadays around the world [1].

Harrison et al. [2] argue how, by building on the advances in IT, the traditional physical city infrastructure is extended to an integrated framework

allowing cities to gather, process, analyze, and make decisions based on detailed operational data. These authors define smart cities through three IT aspects:

- *Instrumented systems* that enable capturing live real-world data describing the operation of both physical and virtual systems of the city (sensors, smartphones, cameras, and social media, among others.)
- *Interconnected systems* enabling the instrumented systems to communicate and interact not only among themselves but also with the multiple IT systems supporting the operation of the city's services.
- *Intelligent systems* able to analyze, model, and visualize the above interconnected data and to derive valuable insights that drive decisions and actions to optimize the operation of the city's services and infrastructure.

Aligned with these aspects, many cities around the world have committed a large amount of resources involving both public and private investment in an effort towards the realization of the smart city vision, yet only few of these initiatives have attained a level of maturity to remain sustainable over time. Research states that one of the key requirements and major challenges for ensuring the sustainability of smart city projects lies in achieving citizen engagement, that is getting communities involved as *prosumers* of the city's data and services [3–5]. This in turn involves providing citizens and other relevant stakeholders with prompt and reliable access to smart city data, enabling them to contribute to the construction and further development of the abovementioned city's *intelligent systems*.

In this context, data management systems are required to handle the massive amounts of data being continuously generated by smart devices. Typically, said data is defined by spatio-temporal dimensions, e.g., weather, air quality, traffic congestion, parking availability, social media streams, etc. Coping with this large volume of spatio-temporal information while supporting time critical end-user applications—such as those enabling responsive data exploration and visualization—is essentially a big data problem that exceeds the ability of traditional offline data processing methods [6, 7]. The nature of this data and requirements of the stated problem call for a more proactive approach where data is *processed during ingestion* in response to recurrent user requests, instead of waiting for it to be accumulated and persisted into an ever-growing database to make it queryable [8, 9].

In that sense, the work reported in this chapter aims at answering the research question on *how to serve common data exploration tasks over live smart city data coming from nonstationary sensors under interactive (low-latency) time constraints.* To address the stated aim, the approach presented

herein explores the use of stream processing over the sequence of readings coming from mobile sensor devices deployed in an urban environment to aggregate the data of those readings into rich summaries for further querying and analysis. The motivation behind this is two-fold:

1. Typical visual exploration applications for this kind of georeferenced time series present users with a sort of dashboard containing a map and a number of controls allowing them to perform visual queries on said data on a per region (e.g., by interacting with the map) and a per time period (e.g., by setting an interval of dates) basis [10]. However, these applications are not able to deliver sensible and predictable response times when operating over highly dynamic data such as the raw readings coming from smart city sensors due to its unbounded size: queries can take from several seconds to minutes over a few million sensor measurements. Considering that these queries define restrictions on the spatial and temporal dimensions of data, it is appealing to establish a fragmentation strategy over these two dimensions in order to reduce the cardinality of the search space by computing continuous data summaries. These summaries amount to a fraction of the number of raw observations, allowing data exploration applications to remain responsive to user queries at the expense of some accuracy.

2. These summaries being proactively derived out of the incoming stream of sensor readings enable data management systems to provide client applications with information about the current state of the measured variables without incurring expensive scan operations over the whole raw data. For said summaries to be relevant, frequent user requests as well as interaction patterns when visually exploring spatio-temporal data should be considered to drive the design of the stream processing pipeline and should determine which technologies could support its operation. By abstracting a generic framework embracing these requirements, it is possible to test to what extent existing data technologies support time-sensitive applications and to estimate their limitations in terms of scalability and reliability.

Aligned with these considerations, the main contributions of the work introduced in this chapter are (*1*) the formulation of a technology-agnostic approach for the continuous computation of data summaries over a live feed of sensor readings by applying a spatio-temporal fragmentation scheme to the sequence of observations, (*2*) the formal definition of a uniform interface for querying said summaries based on recurrent user interaction patterns, and (*3*) the realization of the proposed approach by implementing a complete stream processing pipeline able to operate over real-world sensor readings

coming from a smart city setup deployed in the city of Antwerp in Belgium. For this, a number of existing open source data technologies running on commodity hardware have been used, being able to test their ability to serve visual exploration applications under different configurations. Results show that, by implementing the proposed continuous aggregation approach on centralized and distributed data stores, it is possible to outperform a traditional time series database bringing down query response times by up to two orders of magnitude and reaching sub-second performance for requests made over one year's worth of data (nearly 14+ million observations). This document provides a detailed description of the components and design decisions behind the definition of this approach. Section 3.2 addresses the related work. Section 3.3 focuses on the main contribution of this work and elaborates on the framework for supporting data exploration on spatio-temporal data through continuous computation of data summaries. Section 3.4 describes the implementation of the proposed approach, while Section 3.5 discusses the experimental setup and results. Finally, conclusions and pointers towards future work are provided in Section 3.6.

3.2 Related Work

Recent surveys on big spatio-temporal data by Yang et al. [11] and He et al. [12] argue that most of the existing tools for visual exploration serve a single specific use case, acknowledging the need for more flexible data visualization approaches that allow users to examine the behavioral changes in the information over the temporal and spatial domains while having sensible storage requirements and improving query performance. The approach described in this chapter has been precisely formulated to comply with those requirements, considering smart cities as a meaningful use-case scenario. This section discusses existing literature regarding spatio-temporal data management, visual exploratory analysis on smart city data, and big data frameworks for smart cities.

3.2.1 Spatio-temporal Data Management

The problem of speeding up spatial queries has been studied extensively from the data management perspective. Ganti et al. [13] propose *MP-trie*, a mechanism that reduces the problem of spatial indexing to that of prefix-matching over binary strings by encoding spatio-temporal data into a data structure they call *Space-Time Box* (STB) [14]. According to the authors, *MP-trie* provides a 1000× performance improvement over traditional indexing approaches (specifically *R*-tree* [15]), though it only reaches

said performance when implemented using hardware acceleration (*ternary content-addressable memory* or *TCAM* [16]). *MP-trie* is described as an indexing mechanism intended to speed up spatial queries such as finding all the objects within a distance r from a point p (*range queries*) or finding the *top-K* nearest neighbors from p (*kNN queries*). Similarly, *SATO* [17] and *AQWA* [18] proposed by Vo et al. and Aly et al., respectively, are two data-synopsis-based mechanisms aiming at finding the optimal partitioning scheme in order to lower the response time of spatial queries in distributed spatial datasets. However, these and other similar approaches dealing with spatial indexing and partitioning [19–21] overlook the temporal dimension of the data typical of smart city applications and, in consequence, might fall short in supporting requests intended to explore the historical behaviour from a given sequence of observations.

This issue has also been addressed in the context of *Wireless Sensor Networks* (WSN). Wan et al. [22] present a promising technique for high-dimensional indexing of the sensor data produced within large WSNs, based on the *Voronoi Diagram* data structure. The mechanism that Wan et al. propose includes a hierarchical *in-network* storage which is capable of answering different range monitoring queries, based on the devised indexing scheme. However, given the restrictions in terms of power, storage, and computing resources typical of WSN nodes, pushing a large volume of queries down to the sensing devices for processing could compromise the availability of the network. The approach proposed in this chapter deals with delivering interactive-level performance for basic exploratory tasks. In this use case, it is not uncommon to serve multiple users, each one issuing several queries during a session of data exploration, which would entail a prohibitive computational expense for a WSN.

3.2.2 Visual Exploratory Analysis on Smart City Data

Research on visualization techniques for interactive exploration of smart city data is mainly focused on enhancing user experience by providing them with responsive client-side applications. Doraiswamy et al. [10] proposed *Raster join*, a technique to speed up spatial join queries supporting the interactive exploration of multiple spatio-temporal data sets at the same time. The *Raster join* technique—that leverages current generation graphics processing units (GPU)—was integrated to *Urbane* [23], a 3D visual framework to support the decision making for designing urban development projects. By integrating the proposed technique, *Urbane* is able to handle requests over hundreds of millions of observations with nearly sub-second performance. Similarly, Murshed et al. [24] introduced a web-based application for analysis and visualization spatio-temporal data in smart city applications called

4D CANVAS. This application enables users to perform interactive exploration on both space (*3d*) and time dimensions over a data set stored on disk by leveraging on a WebGL-based framework known as *Cesium* [25]. Also, under these visual data exploration approaches, Li et al. developed *SOVAS* [26], a visual analytics system for query processing of massive array-based climate data, which works on top of *Hadoop* and provides an SQL-based language for users to express their information needs and to conduct spatial analytics tasks. One common feature platforms described in this section (and related solutions like References [27, 28]) that does not incorporate is the ability to process data in a streaming format. These solutions expect the spatio-temporal data they operate on to be residing on the file system (whether local or distributed), some of them requiring additional offline preprocessing to be able to deliver the functionality they advertise.

In contrast to the approaches above, Cao et al. present a visual interactive system known as *Voila* [29], able to process a stream of traffic flow data and to assist users in detecting anomalous events. *Voila* assigns an anomaly score for a given region at a certain point in time by examining changes in patterns' occurrence-likelihoods. Then, users can indicate whether the system has accurately identified anomalous events, and *Voila* incorporates their judgement, recomputing the anomaly scores by using a bayesian approach. In the same vein, Chen et al. proposed *ADF* [30], an open framework for anomaly detection over fine particulate matter measurements (PM2.5), coming from a network of low-cost sensors rolled out on an urban environment. The *ADF* framework is able to identify spatio-temporal anomalous sensor readings as new data comes in, thanks to a statistical-based method called *time-sliced anomaly detection* (TSAD), which thrives on contrasting the readings from each sensing device with those from neighboring sensors to detect and label atypical observations. While the systems proposed by Cao et al. and Cheng et al. were designed with the anomaly-detection use case in mind, the approach described herein was devised for serving a more general purpose, i.e., enabling basic exploratory analysis tasks on live smart city data—regardless of the kind of environmental information being ingested, the number or type of sensor devices, or their location (fixed or mobile)—considering both spatial and temporal data dimensions under interactive response time constraints. It is worth mentioning that one of the main features of the approach introduced in this chapter is that of being an extensible, technology-agnostic data analysis pipeline, and as such, it would be able to integrate the anomaly detection methods implemented in systems like *Voila* and *ADF* while offering interactive querying capabilities over their resulting outcome.

3.2.3 Big Data Frameworks for Smart Cities

As stated earlier, handling spatio-temporal data in the context of smart cities is inherently a big data problem which has become a prolific research field over the last few years. This section addresses some recent advances and initiatives in this regard. Osman A. proposes the *Smart City Data Analytics Panel (SCDAP)* [31], a framework for big data analytics tailored to the specific requirements of smart city environments. *SCDAP* has been laid out in a 3-layered architecture encompassing multiple stages in the data analysis pipeline ranging from data acquisition, cleansing, and transformation to online and batch data processing, including the management and aggregation of data analysis models serving smart city applications. The author outlines a prototype implementation of a big data analytics platform adopting the artifacts defined in *SCDAP*, using a number of existing open source technologies. However, no indication is provided w.r.t. its actual application and performance on real or synthetic smart city data.
Badii et al. [32, 33] introduce *Snap4City*, a visual programming environment along with a suite of microservices allowing users to create event-driven IoT applications in the context of smart cities. The platform runs on top of *Node-RED* [34] and offers a comprehensive set of visual constructs through which users can assemble complex data flows supporting smart city applications (dashboards, route planning, data analytics, etc.). Another platform intended to facilitate the development of smart city applications is *InterSCity* proposed by Del Esposte et al. [35]. *InterSCity* also advocates for a microservice architecture and provides a Web service middleware that enables the integration of heterogeneous IoT devices, services, and resources. While enabling interactive data exploration is not the main concern of platforms like *Snap4City*, *InterSCity*, and other similar approaches [36], their focus on microservices allows for the integration of data management solutions like the one presented in this chapter, aiming at supporting time-sensitive smart city applications.
Aguilera et al. [37] propose *IES Cities*, a data integration platform that enables the creation of citizen-centered applications in the context of smart cities. This approach is founded on the premise that the smart city vision should be achieved through the organic coalescence of government data (*linked open data*), IT infrastructure in place throughout the city (*IoT*), and citizen initiative and contributions mediated through smartphone applications (*crowd-sourced data*). While the *IES Cities* platform is able to integrate smart city data sourced in structured formats such as RDF, JSON, and CSV and relational databases, it does not specifically tackle the issue of enabling interactive data exploration over live streams of spatial-time series data being continuously produced within a smart city environment.

3.3 EXPLORA: Interactive Exploration of Spatio-temporal Data Through Continuous Aggregation

The previous section discussed existing approaches addressing the issue of handling spatio-temporal data to support visual exploratory applications in the context of smart cities. Most of the studies in this review tackle specific aspects of the problem, neglecting in some cases the time dimension of the data; others deal with mechanisms for optimizing display and interaction features, but fall short when processing data as it comes in; and others are concerned with frameworks and guidelines for building smart city applications from the perspective of big data. The proposal addressed in this chapter builds on top of the mentioned approaches and introduces a generic framework called EXPLORA (***E**fficient e**XPLOR**ation through **A**ggregation*) intended for speeding up spatio-temporal queries supporting visual exploratory analysis conducted on mobile sensor data. This section discusses the key requirements and features driving the design of the devised framework, then introduces the enabling techniques adopted to support the framework requirements, elaborates on the framework components and architecture, and finally details the formal definition of the data processing pipeline lying at the core of the framework.

3.3.1 Framework Requirements and Features

User interaction patterns typical in visual data exploration have been identified in a former study by Andrienko et al. [38], distinguishing two main categories of exploratory actions on spatio-temporal data: (*i*) *elementary tasks*, aiming at describing the state of the observed variable(s) at a particular instant (*time*) over a given region (*space*), and (*ii*) *general tasks*, intended for describing how the state of the observed variable(s) in a given region (*space*) changes over *time*. By composing these basic tasks, it is possible to support more elaborate workflows to help answer different questions about the data at hand. This is why the set of categories by Andrienko et al. has become commonplace benchmark tasks to assess the quality of user interactive exploration on spatio-temporal data [39]. On the other hand, a related study by Liu and Heer [40] addressing the effects of latency on visual exploratory analysis states that *high delay reduces the rate at which users make observations, draw generalizations, and generate hypotheses.* Considering these findings, two key requirements have been derived to drive the design of the EXPLORA framework proposed herein:

R3.1 *Support elementary and general visual exploratory tasks on spatio-temporal data generated by mobile sensors in a smart city setup.*

R3.2 *Provide fast answers (sub-second timescales as target) to queries serving the two basic visual exploratory tasks stated in* **R3.1**.

In addition to these key requirements—and following the steps of several of the big data frameworks for smart cities discussed earlier in this document—a microservices approach has been adopted to profit from features such as *modularity*, *extensibility*, and *scalability*. As a generic framework, EXPLORA should be able to incorporate different sources of sensor readings as well as multiple methods for storing, partitioning, and querying said data. Microservices advocate for establishing a clear separation of concerns and for identifying the functional building blocks that support the framework capabilities. This *componentization* facilitates the overall system development and deployment and further promotes other appealing features such as *extensibility* and *maintainability*, reducing the amount of effort required to introduce modifications, since it would involve making said changes to certain individual microservices.

Likewise, an implementation of EXPLORA should be flexible enough to cope with the increasing volumes of sensor data coming in as well as seasonal load variations (e.g., user activity and data influx are expected to peak during certain time periods). The effective modularization into independent deployable components enables these implementations to elastically react to system load; this is, they are able to dynamically *scale-up* or *down* the number of microservice instances they need to efficiently deal with the volume of requests at a given moment.

Lastly, as a consequence of adopting a microservices approach, the EXPLORA framework benefits from two other highly desirable features, namely *availability* and *portability*. By relying on microservices, the framework components are designed to be self-contained and interchangeable, which helps in timely spotting system failures when they occur, introducing changes to the relevant components and redeploying them without incurring in any major system downtime. Microservices also encourage the use of well-defined interfaces exposing the capabilities of each component and mediating the interaction with other system modules and the underlying infrastructure. This way, as long as modules comply to said interfaces, details such as the language they are written in and the software frameworks they use are not relevant.

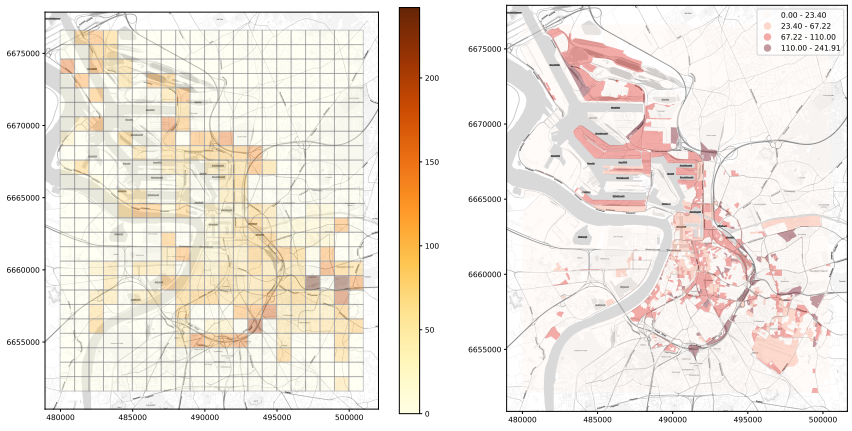
3.3.2 Enabling Techniques

To comply to the committed requirements, the EXPLORA framework relies on two enabling techniques: query categorization and data synopsis.

3.3.2.1 Query Categorization

Requests serving *elementary* and *general* exploratory tasks (requirement **R3.1**) query spatio-temporal data on different attributes and satisfy different information needs. When conducting *elementary tasks*, users are interested in visualizing the state of the observed variable over a particular region at a given moment in time. For instance, a user might want to know the concentration of particulate matter (PM) over their neighbourhood during peak hours. Queries serving these kind of tasks expect the requested time (in terms of *timestamps*) and the geographic area of observation (in terms of *longitude* and *latitude*) as input parameters and provide as output a sort of snapshot accounting for the value of the observations aggregated over discretized units of space covering the region of interest. Typical examples of the kind of visualizations that might be presented to the user as a result of these *elementary* exploratory tasks are the choropleth maps shown in Figure 3.1. Queries falling into this category have been labeled as *Snapshot-temporal queries* (ST).

Figure 3.1 Examples of visualizations expected from an *elementary* exploratory task: These choropleth maps show the concentration of nitrogen dioxide (NO_2) over the city of Antwerp, BE through a one-month period.

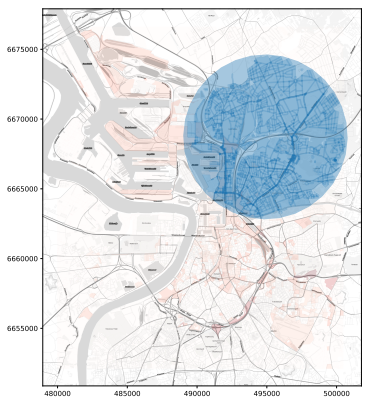


(a) Tile-based choropleth

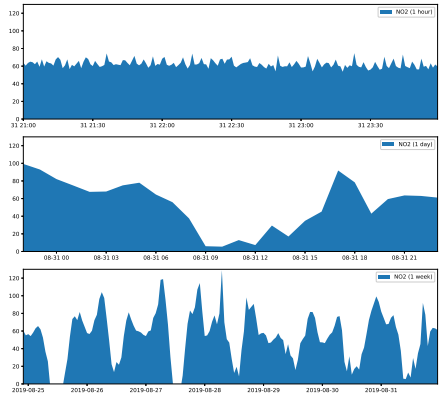
(b) Street-block-based choropleth

On the other hand, the intent behind *general* exploratory tasks consists in comparing the state of the observed variable over a given region along different points in time. Users might conduct these kind of tasks, for instance, by selecting an arbitrary geographic area on a map and by choosing the period of time they are interested in reviewing. This way, queries supporting *general* exploratory tasks expect as inputs a specification of the region of interest along with the inspection time period and yield as answer the value of the observed variable aggregated over discretized units of time (minutes, hours, days, etc.), revealing the historical behaviour of the measured variable. Queries belonging to this category are referred to as *Historical-spatial queries* (HS). Figure 3.2 below outlines a common outcome of a *general* exploratory task.

Figure 3.2 Example of a *general* exploratory task: (**a**) the visual query prompted by the user: *How have the NO_2 emissions historically evolved within the traced perimeter?* (**b**) Three time series charts reporting on the concentration of the NO_2 over the past one hour, 24 h, and one week.



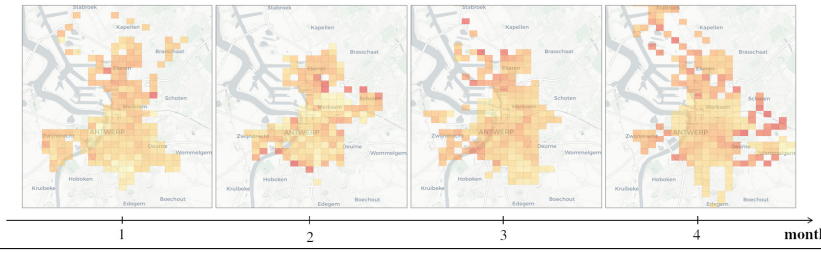
(a) Polygonal selection from a 2D map.



(b) Time series corresponding to the selected region.

It is worth noting that a *general* exploratory task can be fulfilled as well by sequentially executing multiple *elementary* tasks. Consider for instance the case in Figure 3.3, where a progression of choropleths is displayed, as the result of running a series of *snapshot-temporal* queries requesting the state of the observed variable over several months. While in practice this sequence of requests serve a *general task* intent, in cases like this, the proposed framework deals with each individual query in isolation, regardless of the overall purpose of the exploratory task.

Figure 3.3 Example of a *general* exploratory task as a composite of multiple *elementary* tasks.



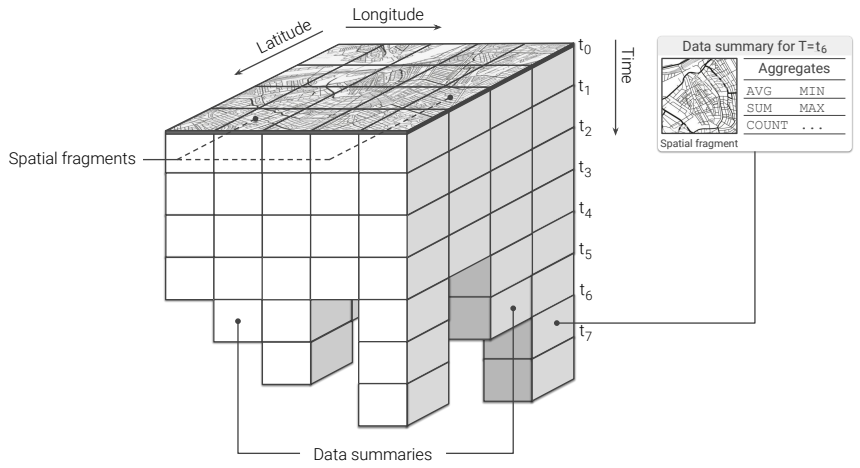
3.3.2.2 Data Synopsis and Spatio-temporal Fragmentation

Both *historical-spatial* and *snapshot-temporal* queries are expensive and time-consuming when running on large spatio-temporal data, since they involve executing demanding scan, sort, and aggregate operations. Reducing the data-to-insight time in visual exploratory applications requires speeding up this kind of query (requirement **R3.2**). The temporal and spatial dimensions of smart city data along with the specific features of HS and ST queries make this problem appealing for synopsis data structures [41]. Synopsis structures are by definition substantially smaller than the base data set they are derived from. They represent a summarized view of the original data intended to serve certain predefined types of queries. In a streaming setting, synopsis structures are created as data comes in; this way, users can submit queries on the data stream at any point in time and get prompt (and often approximate) answers based only on the data available thus far in the synopsis structures.

As stated in the previous section, the outcome of queries supporting visual exploratory applications is typically delivered in discretized units of time (HS queries) or space (ST queries). EXPLORA takes advantage of such discretization to assemble synopsis structures—namely, *continuous views*—that are incrementally computed as new sensor observations arrive. Consider the choropleth maps presented back in Figure 3.1, reporting on the concentration of nitrogen dioxide (NO_2) over the city of Antwerp, Belgium, for a period of one month. To build these visualizations, raw sensor readings occurring during the requested period are aggregated according to the *spatial fragment* (i.e., tile/street-block) which they fall into. Then, the value of said aggregate is encoded in the color displayed for each fragment, providing the user with insight about the state of the observed variable. Similarly, the time series charts shown in Figure 3.2b are laid out by aggregating raw observations into specific time resolutions or bins (i.e., minutes, hours, and days) in correspondence to the time said observations occurred. Instead of

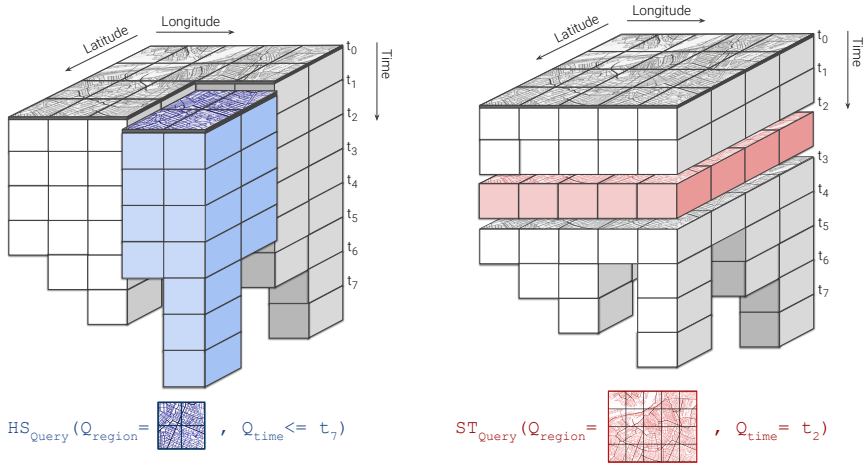
computing these aggregates on request over the raw sensor observations, EXPLORA sets a spatial fragmentation scheme upfront and applies multiple aggregate operations (e.g., *average*, *sum*, and *count*) for a number of time resolutions (from one-minute to monthly) over the incoming stream of sensor readings. The collection of aggregates corresponding to an individual spatial fragment over a single time bin has been labeled as *data summary*. This way, continuous views are assembled for each of the supported time resolutions by persisting the resulting data summaries into an structure that can be seen as a sort of *dynamic spatio-temporal raster*. Figure 3.4 below shows a schematics of this structure, in which a regular tile grid is used as spatial fragmentation strategy for illustrative purposes.

Figure 3.4 Spatio-temporal fragmentation for continuous computing of data summaries.



Notice that, regardless of the volume of sensor readings being ingested, the size of the continuous views only depends on the size of the spatial fragments and time bins being used, this is, the lower the resolution of the spatio-temporal fragmentation scheme, the smaller the size of the corresponding view. By querying these synopsis structures instead of the raw sensor data, users of visual exploratory applications can experience a more responsive feedback at the expense of some accuracy. It is worth noting as well how, thanks to the way these continuous views are structured, answering HS and ST queries comes down to cutting longitudinal (i.e., along the *time* axis) and transverse slices (i.e., along the *longitude/latitude* plane), respectively, and further aggregates their constituent data summaries afterwards, as illustrated below in Figure 3.5.

Figure 3.5 Query resolution on continuous views: The diagram on the left describes an HS query requesting the historical behaviour of the observed variable over Q_{region}, while the one to the right shows an ST query requesting the state of the observed variable at instant $Q_{time} = t_2$.

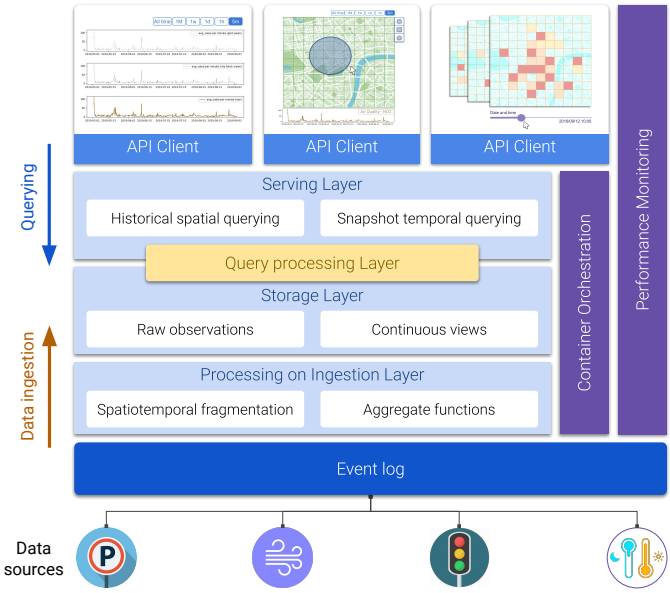


3.3.3 The EXPLORA Framework: Components and Architecture

This section deals with the definition of the framework's building blocks and how they fit together to meet the requirements stated earlier. As Figure 3.6 illustrates, the EXPLORA framework adopts a layered architecture approach, where functional modules are organized into logical tiers, namely *processing on ingestion*, *storage*, *query processing*, and *serving* layers. Besides these functional layers, three supporting layers are defined for decoupling the system from the available sensor data sources (*event log*) and for providing monitoring capabilities and infrastructure resources for the components in the functional tiers to operate with (*performance monitoring* and *container orchestration*). The description of these layers and their associated components is addressed next.

Event log This layer serves as an interface between the framework and the sensor data providers. It collects the raw sensor data and hands it over to the upper layers for scalable and reliable consumption and further processing. This tier can be realized through a distributed append-only log that implements a *publish–subscribe* pattern, allowing data producers to post raw sensor observations to logical channels (topics) that are eventually consumed by client applications in an asynchronous way.

Figure 3.6 Components and architecture of the EXPLORA framework.



Processing on ingestion This layer subscribes to the *event log* to consume the stream of raw sensor observations and processes them to continuously generate the data synopsis structures that the framework thrives on. The stream processing mechanism this layer implements is subject to the particular designated *spatio-temporal fragmentation* strategy and the set of supported *aggregate functions* used to compute the corresponding data summaries. This layer represents one of the core components of the EXPLORA framework, as it comprises the modules in charge of applying the ingestion procedure that will be further discussed later in this section (Algorithm 3.1).

Storage layer This tier comprises the artifacts responsible for providing persistent storage for both the *continuous views* generated in the ingestion layer and the stream of *raw sensor observations* being consumed from the *event log*, along with the corresponding programming interfaces (APIs) for enabling modules in adjacent tiers to conduct basic data retrieval tasks. Complex requests—such as those supporting the *elementary* and *general* exploratory tasks discussed back in Section 3.3.1—might be handled in cooperation with the *serving layer* at the top, depending on querying capabilities offered by the data storage technologies implemented in this layer.

Serving layer This tier provides an entry point for visual exploratory applications to interact with the framework and to access the available sensor data. The serving layer implements a uniform API allowing client applications to issue *historical-spatial* and *snapshot-temporal* queries against the data persisted in the storage layer (both raw observations and continuous views). Depending on the storage technologies used in the underlying *storage layer*, the serving tier might also take part in the query resolution process. This is why *query processing* is represented as a separate layer, sitting in between the two upper tiers.

Query processing As stated above, responsibilities of this tier overlap those from the contiguous layers (*serving* and *storage*). The processing performed in this layer supports query answering for both *historical-spatial* and *snapshot-temporal* inquiries (according to the procedures detailed in Algorithms 3.2 and 3.3, discussed later in Section 3.3.4). Where this processing takes place is determined by the capabilities of query API provided by the data storage being used. Thus, for instance, a data store offering an expressive SQL interface would be able to handle most of the query processing tasks, while a typical *key-value* store offering simple lookup operations would require a large part of the query processing to be performed programmatically in the *serving layer*.

Container orchestration All the functional components of the EXPLORA framework are implemented as containerized microservices. The *container orchestration layer* is in charge of the automatic deployment, scaling, load balancing, networking, and life-cycle management of the containers that these components operate on. Examples of existing technologies able to support the functionality required from this layer are *Kubernetes* [42]—deemed as the *de facto* standard for container orchestration to date—*OpenShift* [43], and *Apache Mesos* [44].

Performance monitoring The role of this layer is to keep track of a number of metrics accounting for the computing requirements (memory and CPU usage) and overall performance of a system implementing the EXPLORA framework (query response time and accuracy). To that end, this layer relies on tools provided by the *container orchestrator*, the operating system, and third-party libraries for statistical analysis and data visualization. Performance information such as that reported later in Section 3.5 is compiled in this layer.

Client applications Finally, visual exploratory applications consume the API available through the *serving layer* to support different data exploration use cases based on the two abstracted categories of exploratory tasks: *elementary* and *general*. Section 3.4 provides a number of examples of said use cases, presented as part of proof-of-concept implementations of the proposed framework.

3.3.4 The EXPLORA Framework: Formal Methods and Algorithms

The formal definition of the query resolution mechanism along with the ingestion procedure at the core of the EXPLORA framework are detailed next.

3.3.4.1 Data Ingestion: Continuous Computation of Data Synopsis Structures

Let us represent a mobile sensor observation (reading) as the following tuple:

$$r = \langle t, x, y, s, v, a_0, a_1, \ldots, a_n \rangle \tag{3.1}$$

where t is the *timestamp* indicating when the observation was made, x and y being respectively the *longitude* and *latitude* where the observation took place; s is the *observed (sensed) variable*; v is the *observed value* as measured by the sensor; and a_i is additional attributes and metadata (device identifier, measurement units, etc.).
Then, a continuous view for a given observed variable, s, can be represented as a function $\mathcal{V}$ that maps a spatial fragment, ϕ, and a temporal bin, τ, to its corresponding data summary $\boldsymbol{\sigma}$ (collection of aggregates), as follows:

$$\mathcal{V}_{\langle s, \boldsymbol{\Phi}, \Omega \rangle} : (\phi, \tau) \mapsto \boldsymbol{\sigma}; \quad \boldsymbol{\sigma} = \{\sigma_{\texttt{AVG}}, \sigma_{\texttt{SUM}}, \sigma_{\texttt{COUNT}}, \ldots\} \tag{3.2}$$

where ϕ is one of the discretized units of space into which a geographic area is partitioned, according to a certain spatial fragmentation strategy $\boldsymbol{\Phi}$ (e.g., tiles, hexagons, street blocks, etc.) and τ identifies one of the temporal buckets resulting from setting a regular frequency, Ω, at which the incoming sensor observations are aggregated (e.g., minutely, hourly, daily, etc.).
Similarly, the mechanism for assigning a sensor observation r to its corresponding data summary can be defined as a function $\mathcal{F}$ that takes the spatial and temporal attributes from r and returns the spatial fragment and temporal bin identifying the data summary to which r belongs:

$$\mathcal{F} : r\langle t, x, y \rangle \mapsto (\phi, \tau) \tag{3.3}$$

With these definitions in place, the formal procedure for data ingestion in EXPLORA is presented below in Algorithm 3.1. The process starts by first setting a spatial fragmentation strategy ($\mathbf{\Phi}_k$), a frequency of aggregation (Ω_k), and a set of aggregate methods to be supported ($\mathbf{\Sigma}_k$) (lines 3–6). Then, persistent storage for a new continuous view is allocated (assuming it does not exist yet) (line 7), and the sensor observations coming from a stream $\boldsymbol{\mathcal{S}}$ are taken in, one after the other. To determine the data summary into which each sensor reading has to be aggregated, the spatial fragment and temporal bin are computed by applying the function $\mathcal{F}$ on each of the incoming readings. With this input, the corresponding data summary is retrieved from the view (lines 9 and 10). Then, the collection of aggregates from the data summary gets updated and the changes are persisted in the continuous view (lines 11–19). In practice, the type of each one of the aggregate functions in $\mathbf{\Sigma}_k$ (i.e., *distributive*, *algebraic*, or *holistic* [45]) determines how the update procedure in line 16 is implemented. Later, in Section 3.4, two prototypes are presented for illustration and proof of concept.

Algorithm 3.1 EXPLORA ingestion procedure.

1: Let $\boldsymbol{\mathcal{S}}$ be a stream of sensor observations of a variable $\hat{s}$
2: $\boldsymbol{\mathcal{S}} = \{r_0, r_1, r_2, \dots\}$; $r_i = \langle t_i, x_i, y_i, \hat{s}, v_i, a_{0i}, a_{1i}, a_{2i}, \dots \rangle$ ▷ Unbounded set of sensor readings
3: Let $\mathbf{\Phi}_k$ be a spatial fragmentation strategy
4: $\mathbf{\Phi}_k = \{\phi_0, \phi_1, \phi_2, \dots, \phi_n\}$
5: Let Ω_k be the frequency of aggregation ▷ e.g. minutely, hourly, daily
6: Let $\mathbf{\Sigma}_k$ be a set of aggregate operations ▷ e.g. `AVG`, `SUM`, `COUNT`
7: Create persistent storage for view $\mathcal{V}_{\langle \hat{s}, \mathbf{\Phi}_k, \Omega_k \rangle}$
8: **for each** reading r_i in $\boldsymbol{\mathcal{S}}$ **do**
9: $(\phi_i, \tau_i) \leftarrow \mathcal{F}(r_i\langle t_i, x_i, y_i \rangle)$; $\phi_i \in \mathbf{\Phi}_k$ ▷ Get the spatial fragment and temporal bin for r_i
10: $\boldsymbol{\sigma_i} \leftarrow \mathcal{V}_{\langle \hat{s}, \mathbf{\Phi}_k, \Omega_k \rangle}(\phi_i, \tau_i)$ ▷ Get the data summary r_i should be aggregated into
11: **for each** operation `AGGR` in $\mathbf{\Sigma}_k$ **do** ▷ Update data summary aggregates
12: $\sigma_{\texttt{AGGR}} \leftarrow \boldsymbol{\sigma_i}$ [`AGGR`]
13: **if** $\sigma_{\texttt{AGGR}} = \emptyset$ **then** ▷ If there is no aggregate for `AGGR` yet, then initialize it with r_i
14: $\sigma_{\texttt{AGGR}} \leftarrow$ `AGGR`$(r_i\langle v_i \rangle)$
15: **else** ▷ Otherwise, update the current aggregate for `AGGR` with r_i
16: Update $\sigma_{\texttt{AGGR}}$ with $r_i\langle v_i \rangle$
17: **end if**
18: Update $\sigma_{\texttt{AGGR}}$ in $\boldsymbol{\sigma_i}$
19: Persist $\boldsymbol{\sigma_i}$ in $\mathcal{V}_{\langle \hat{s}, \mathbf{\Phi}_k, \Omega_k \rangle}$ ▷ Finally, update the continuous view
20: **end for**
21: **end for**

As soon as sensor observations start being ingested, EXPLORA is capable of processing queries issued against the continuous views. The mechanism for query resolution varies from HS queries to ST queries. The subsections below detail the procedure for each category of queries, starting by defining their corresponding functions.

3.3.4.2 Query Processing: *Historical-Spatial Queries*

Let us define $\mathcal{HS}$—for *historical-spatial* queries—as a function that takes as inputs a specification of an arbitrary polygonal selection, $\boldsymbol{\phi_q}$, from a 2-dimensional map (e.g., as an array of vertex coordinates) and optionally an interval of dates, $\tau_{q\langle \texttt{start},\texttt{end}\rangle}$, and delivers as output an array containing the data summaries aggregated over all the spatial fragments ϕ_i lying inside the perimeter defined by $\boldsymbol{\phi_q}$, for all the temporal bins τ_i in τ_q:

$$\begin{aligned}\mathcal{HS}\Big|_{\mathcal{V}_{\langle s,\boldsymbol{\Phi},\Omega\rangle}} : (\boldsymbol{\phi_q}, \tau_{q\langle \texttt{start},\texttt{end}\rangle}) \mapsto \{&\langle \tau_m, \boldsymbol{\sigma_m}\rangle, \langle \tau_{m+1}, \boldsymbol{\sigma_{m+1}}\rangle, \\ &\langle \tau_{m+2}, \boldsymbol{\sigma_{m+2}}\rangle, \ldots, \langle \tau_n, \boldsymbol{\sigma_n}\rangle\}; \\ \tau_m \geq \tau_{q\langle \texttt{start}\rangle} \wedge \tau_n \leq \tau_{q\langle \texttt{end}\rangle}&\end{aligned} \tag{3.4}$$

where $\mathcal{V}_{\langle s,\boldsymbol{\Phi},\Omega\rangle}$ is the continuous view that the $\mathcal{HS}$ function is evaluated against and each $\boldsymbol{\sigma_k}$ is the aggregated summary that results from combining the data summaries corresponding to the spatial fragments covered by $\boldsymbol{\phi_q}$, for temporal bin τ_k. The procedure for deriving said aggregated summaries is formally defined below in Algorithm 3.2.

Algorithm 3.2 EXPLORA query processing for *historical-spatial* queries.

Let $\mathcal{V}_{\langle \hat{s},\boldsymbol{\Phi}_k,\Omega_k\rangle}$ be a continuous view being fed with sensor observations from a variable $\hat{s}$, with spatial fragmentation $\boldsymbol{\Phi}_k$ and aggregation frequency Ω_k
Let $\boldsymbol{\Sigma}_k$ be a set of aggregate operations ▷ e.g. `AVG`, `SUM`, `COUNT`
procedure $\mathcal{HS}(\boldsymbol{\phi_q}, \tau_{q\langle \texttt{start},\texttt{end}\rangle})$
    input: an arbitraty polygonal selection from a 2D map ($\boldsymbol{\phi_q}$) and a time interval ($\tau_{q\langle \texttt{start},\texttt{end}\rangle}$)
    output: summary time-series ($\boldsymbol{\mathcal{R}_{\mathcal{HS}}}$)
    $\boldsymbol{\Phi_q} \leftarrow \boldsymbol{\Phi}_k \cap \boldsymbol{\phi_q}$ ▷ Get the set of spatial fragments inside $\boldsymbol{\phi_q}$
    $\tau_m \leftarrow \texttt{truncate}_{\Omega_k}(\tau_{q\langle \texttt{start}\rangle})$ ▷ Starting temporal bin
    $\tau_n \leftarrow \texttt{truncate}_{\Omega_k}(\tau_{q\langle \texttt{end}\rangle})$ ▷ Ending temporal bin
    $\boldsymbol{\mathcal{R}_{\mathcal{HS}}} \leftarrow \{\}$ ▷ Initialize result set
     for $\tau_i = \tau_m$ **to** τ_n **do**
        $\boldsymbol{\sigma_i} \leftarrow \{\}$ ▷ Initialize empty aggregated summary for τ_i
        for each fragment ϕ_j in $\boldsymbol{\Phi_q}$ **do**
            $\boldsymbol{\sigma_j} \leftarrow \mathcal{V}_{\langle \hat{s},\boldsymbol{\Phi}_k,\Omega_k\rangle}(\phi_j, \tau_i)$ ▷ Get the data summary for ϕ_j and τ_i
            for each operation `AGGR` in $\boldsymbol{\Sigma}_k$ **do**
                $\sigma_{\texttt{AGGR}} \leftarrow \boldsymbol{\sigma_i}$ [`AGGR`]
                if $\sigma_{\texttt{AGGR}} = \emptyset$ **then** ▷ If there is no aggregate for `AGGR` yet, then initialize it with $\boldsymbol{\sigma_j}$
                    $\sigma_{\texttt{AGGR}} \leftarrow \boldsymbol{\sigma_j}$ [`AGGR`]
                else ▷ Otherwise, combine the existing aggregate for `AGGR` with $\boldsymbol{\sigma_j}$
                    Combine $\sigma_{\texttt{AGGR}}$ with $\boldsymbol{\sigma_j}$ [`AGGR`]
                end if
                Update $\sigma_{\texttt{AGGR}}$ in $\boldsymbol{\sigma_i}$ ▷ Update the aggregated summary $\boldsymbol{\sigma_i}$
            end for
        end for
        Append $\langle \tau_i, \boldsymbol{\sigma_i}\rangle$ to $\boldsymbol{\mathcal{R}_{\mathcal{HS}}}$
     end for
     return $\boldsymbol{\mathcal{R}_{\mathcal{HS}}}$ ▷ The time series of aggregated summaries
end procedure

First, the set of spatial fragments $\boldsymbol{\Phi_q}$ lying inside $\boldsymbol{\phi_q}$ is computed (this operation has been represented as the *set intersection* in line 6). Then, the boundary temporal bins, τ_m and τ_n, are defined by truncating the $\tau_{q\langle\texttt{start}\rangle}$ and $\tau_{q\langle\texttt{end}\rangle}$ dates, respectively, according to the frequency of aggregation Ω_k (lines 7 and 8). This is, for instance, if Ω_k is set to *hourly*, then the dates are truncated to the exact hour (e.g., `2019-09-22T12:47:32.767Z` → `2019-09-22T12:00:00.000Z`). Once these time boundaries have been determined, the data summaries corresponding to the fragments in $\boldsymbol{\Phi_q}$ are retrieved from the view and aggregated for each of the temporal bins in the interval $[\tau_m, \tau_n]$ (lines 11–21). Finally, the resulting aggregated summaries, along with their corresponding temporal bins, are paired together and incrementally appended to the result set to assemble the summary time series returned as output ($\boldsymbol{\mathcal{R}_{\mathcal{HS}}}$) (lines 24–26).

3.3.4.3 Query Processing: *Snapshot-Temporal Queries*

On the other hand, $\mathcal{ST}$—for *snapshot-temporal* queries—is a function that takes as inputs the timestamp τ_q at which a snapshot of the state of the observed variable would be taken and optionally a polygonal selection, $\boldsymbol{\phi_q}$, from a 2-dimensional map (if not provided, the snapshot would be computed over the entire region for which data is available). With these inputs, $\mathcal{ST}$ returns the collection of data summaries that correspond to the spatial fragments lying inside of $\boldsymbol{\phi_q}$ (if provided) for the temporal bin τ_x, where τ_q falls into:

$$\mathcal{ST}\Big|_{\mathcal{V}_{\langle s,\boldsymbol{\Phi},\Omega\rangle}} : (\tau_q, \boldsymbol{\phi_q}) \mapsto \{\langle \phi_a, \boldsymbol{\sigma_a}\rangle, \langle \phi_b, \boldsymbol{\sigma_b}\rangle, \langle \phi_c, \boldsymbol{\sigma_c}\rangle, \dots\}; \quad \phi_x \in \boldsymbol{\Phi} \cap \boldsymbol{\phi_q} \tag{3.5}$$

where $\mathcal{V}_{\langle s,\boldsymbol{\Phi},\Omega\rangle}$ is the continuous view that the $\mathcal{ST}$ function is evaluated against and each $\boldsymbol{\sigma_x}$ is a data summary registered under the temporal bin τ_x (namely, the one τ_q fits into). The generic sequence of steps followed by this function is detailed below in Algorithm 3.3. The procedure is similar to the one defined for the $\mathcal{HS}$ function. It also starts by computing the set of spatial fragments lying under the selected polygonal region and by determining τ_x from the provided timestamp (τ_q) by applying a `truncate` operation (lines 5 and 6). Then, the data summaries available in the view are filtered, so that only those corresponding to the spatial fragments covered by ϕ_q and registered under τ_x are retrieved (lines 8–14). These summaries and their corresponding spatial fragments are coupled together and appended to a collection of tuples ($\boldsymbol{\mathcal{R}_{\mathcal{ST}}}$), representing a temporal snapshot of the observed variable.

Algorithm 3.3 EXPLORA query processing for *snapshot-temporal* queries.

Let $\mathcal{V}_{\langle \hat{s}, \mathbf{\Phi}_k, \Omega_k \rangle}$ be a continuous view being fed with sensor observations from a variable $\hat{s}$, with spatial fragmentation $\mathbf{\Phi}_k$ and aggregation frequency Ω_k
procedure $\mathcal{TS}(\tau_q, \boldsymbol{\phi_q})$
  input: a snapshot timestamp (τ_q) and a polygonal selection from a 2D map ($\boldsymbol{\phi_q}$)
  output: temporal snapshot ($\boldsymbol{\mathcal{R}}_{\mathcal{ST}}$)
  $\mathbf{\Phi_q} \leftarrow \mathbf{\Phi}_k \cap \boldsymbol{\phi_q}$ ▷ Get the set of spatial fragments inside $\boldsymbol{\phi_q}$
  $\tau_x \leftarrow \texttt{truncate}_{\Omega_k}(\tau_q)$ ▷ Get the querying temporal bin
  $\boldsymbol{\mathcal{R}}_{\mathcal{ST}} \leftarrow \{\}$ ▷ Initialize result set
  for each fragment ϕ_x in $\mathbf{\Phi_q}$ **do**
    $\boldsymbol{\sigma_x} \leftarrow \mathcal{V}_{\langle \hat{s}, \mathbf{\Phi}_k, \Omega_k \rangle}(\phi_x, \tau_x)$ ▷ Get the data summary for ϕ_x and τ_x
    if $\boldsymbol{\sigma_x} \neq \emptyset$ **then** ▷ If there is a data summary under (ϕ_x, τ_x)
      Append $\langle \phi_x, \boldsymbol{\sigma_x} \rangle$ to $\boldsymbol{\mathcal{R}}_{\mathcal{ST}}$
    end if
  end for
  return $\boldsymbol{\mathcal{R}}_{\mathcal{ST}}$ ▷ Snapshot of $\hat{s}$, over $\boldsymbol{\phi_q}$ at τ_x
end procedure

The three algorithms formulated in this section lie at the core of EXPLORA, allowing for interactive exploration of mobile sensor data. It is worth noting that, since these algorithms operate on discretized units of space and time, in most of the cases, they would only manage to deliver approximate query answers; this is, the gain in speed this framework brings in entails a loss in accuracy. Nevertheless, for use cases in visual exploratory analysis, these estimates are able to provide relevant insights on the state and historical behaviour of the observed variables. Later, in Section 3.5, a metric is introduced to measure accuracy of queries issued against continuous views—under several spatio-temporal fragmentation strategies—w.r.t. queries running on the base raw data.

To recap, this section developed a thorough description of the framework devised for enabling interactive exploration of mobile sensor data in smart cities. It started by identifying the framework requirements and features. Then, a description of the key techniques behind the formulation of the proposed framework was discussed. Next, the definition of the layered architecture adopted for the proposed framework along with the description of its constituent modules were addressed, and finally, a comprehensive presentation of the mechanisms behind the stream processing pipeline that enables the continuous generation of data synopsis structures as well as the procedures defined to speed up spatio-temporal queries, which profit from said data synopsis structures, were made.

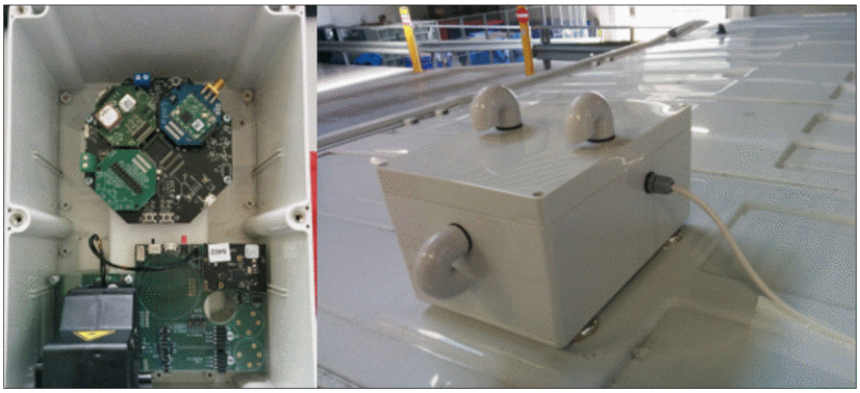
3.4 Prototype Implementation

This section explores the applicability of the EXPLORA framework for enabling interactive exploration of live and historical smart city data by harnessing existing open source data technologies. First, an application sce-

nario within the context of mobile sensor data in smart cities is described. Then, three target use cases of visual exploratory applications are defined, incorporating the *elementary* and *general* exploratory tasks identified in the previous section to provide more elaborate interaction workflows. Lastly, two implementations aimed at supporting the defined use cases are detailed, one based on a traditional spatial time-series database approach and another using a distributed stream processing approach.

3.4.1 Application Scenario: The Bel-Air Project

Figure 3.7 Setup of air quality sensors installed on the roofs of the *Bpost* delivery vans as part of the *Bel-Air* project.



The *Bel-Air* project is part of the *City of Things* (CoT) [46] initiative that is being implemented in the city of Antwerp, Belgium, in a joint effort that involves businesses, government, and academia. This initiative aims at putting together a city living lab and technical testbed environment, which allows researchers and developers to easily set up and validate IoT experiments. Within CoT, the Bel-Air project is particularly concerned with finding efficient mechanisms to accurately measure the air quality over the city. Since the costs of rolling out a dense network of fixed sensors across a large urban area could be prohibitively expensive, the Bel-Air project established a partnership with the Belgian Postal service (*Bpost*) to attach highly sensitive sensors to the roofs of the mail delivery vans that traverse the city on a daily basis (see Figure 3.7). These sensors conduct periodic measurements on environmental variables such as temperature, humidity, and air pollution (particulate matter, nitrogen dioxide, etc.), which are timestamped and geotagged before being sent over the network to a persistent storage. This

mobile sensor setup together with some additional sensors deployed at fixed locations allow mapping the air quality of the entire city of Antwerp in a cost-effective way.

Efficient mechanisms for visual exploratory analysis over the data delivered by the mobile-sensor setup of the Bel-Air project can help get relevant insights regarding the status of air quality across the urban area of Antwerp, which would further allow to timely take the proper course of action to mitigate the problems caused by elevated levels of pollution. This scenario serves as the context for a proof-of-concept realization for the proposed framework. The next section describes a number of target use cases to test the applicability of the EXPLORA approach.

3.4.2 Target Use Cases for Visual Exploratory Applications on Spatio-temporal Data

3.4.2.1 Visualizing the Temporal Change of an Observed Variable over a Certain Region

This use case has to do with allowing users to pose visual queries aimed at examining the historical behaviour of an air quality variable by defining a polygonal selection on a 2-dimensional map. Queries are further parameterized, allowing users to specify traits such as the *aggregate function* they want to be applied on the data, the *time resolution* (per-minute, per-hour, or per-day) or *time period* (last 5 minutes, last hour, etc.) they want the results to be displayed on, and whether the query should be issued against the raw sensor data or run against the continuous views computed during data ingestion. Figure 3.2 (in Section 3.3.2.1) and Figure 3.8 below portray examples of this use case.

3.4.2.2 Progressive Approximate Query Answering

Aiming at improving the user experience in terms of perceived responsiveness, queries supporting visual exploratory actions can profit from the reduced latency expected from synopsis data structures. In this sense, users can be presented first with an approximate answer to their requests, which then is continuously refined as time goes on, until the exact result—computed on the raw data—is finally displayed. To this end, multiple continuous views are required to be computed during data ingestion, featuring progressively finer geospatial resolution.

Figure 3.8 Application allowing users to examine the change over time of an air quality measure on a certain geospatial region: Notice at the right hand side how tiles and street-blocks would approximate the area of the provided polygonal selection.

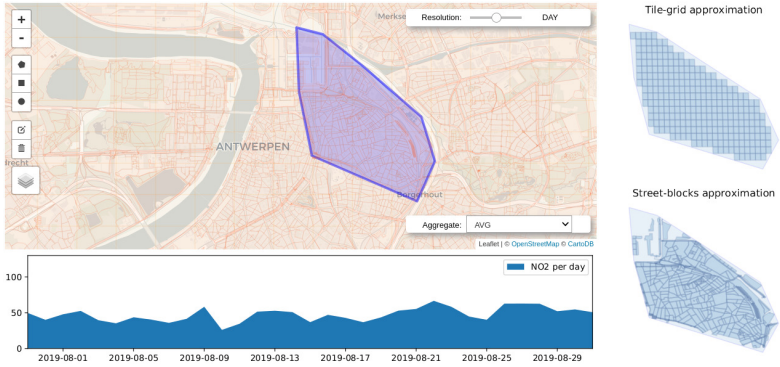
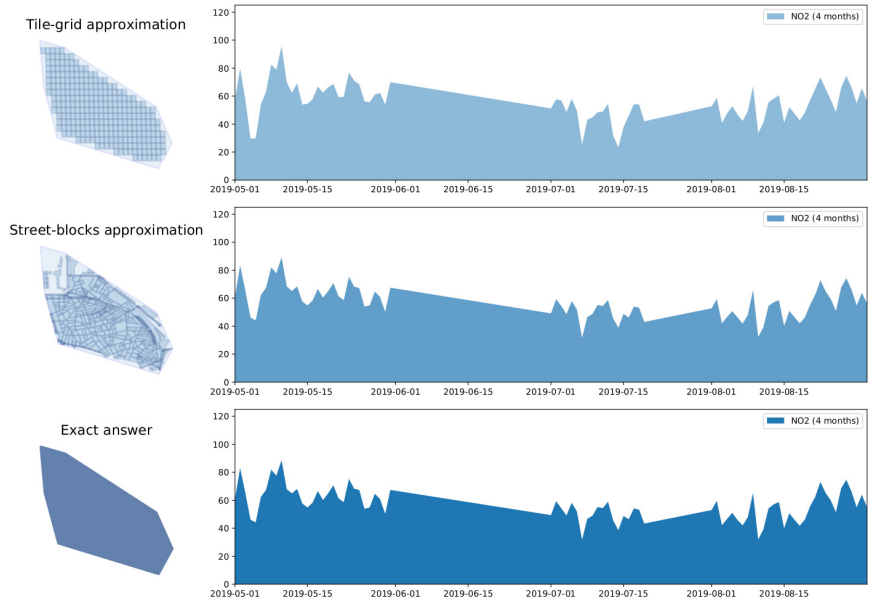


Figure 3.9 Progressive approximate query answering: The approximate time series at the top gets gradually refined until the exact answer is presented to the user.



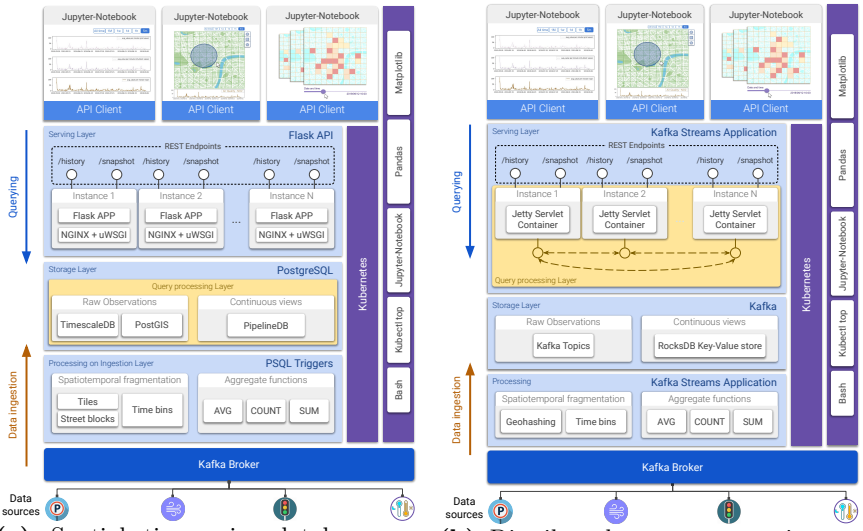
Consider for instance the time series charts in Figure 3.9, corresponding to the polygonal selection in Figure 3.8 over a period of 4 months. Notice how the resulting time series is progressively refined from the chart at the top to the one at the bottom, which corresponds to the final exact answer derived from the raw sensor observations.

3.4.2.3 Dynamic Choropleth Map

This use case concerns the visualization of the historical behaviour of a given variable, this time by displaying a sequence of successive temporal snapshots and by allowing the user to transition between them on command using interactive controls (e.g., back and forward buttons or a time slider). An example of this use case was presented earlier in Figure 3.3 when discussing the execution of a *general* exploratory task as a composite of multiple *elementary* tasks.

3.4.3 Proof-of-Concept Implementations of EXPLORA

Figure 3.10 Proof-of-concept implementations of EXPLORA.



(a) Spatial time-series database approach.

(b) Distributed stream processing approach.

This section describes two realizations of the *Explora* framework: the first one harnesses a series of extensions of the *PostgreSQL* open-source relational database management system (RDBMS), which endow this database engine

with capabilities for efficiently storing and indexing time-series and geospatial data. The second implementation draws on the distributed stream processing engine provided by Apache Kafka [47] to process the feed of sensor observations from the Bel-Air project setup. Figure 3.10 presents two diagrams mapping the technologies used in both implementations to each of the tiers and components of the EXPLORA framework. Let us first consider the modules which are common to both implementations and then proceed to a detailed description of those that are specific to each approach.

Event log *Apache Kafka* is used to implement this layer of the architecture. Kafka provides a number of tools for processing and analyzing streams of data, including a distributed message broker that adopts the *publish–subscribe* pattern. This Kafka broker allows for registering each of the incoming sensor observations into a partitioned append-only log, maintaining them over a fixed configurable retention period, which enable multiple consumers (as many as the number of partitions) to read and process the collected data in an asynchronous-concurrent way.

Container orchestration The components in the *serving*, *storage*, and *processing on ingestion* layers are built as Docker containers and are deployed on a Kubernetes cluster, consisting of one master node and three working nodes, all of them running *Ubuntu 18.04.3 LTS*.

Performance monitoring Data regarding query response time, query accuracy, and computing resources usage for all the components of the system is captured via bash and Python scripting. Once collected, this information is analyzed and visualized through a series of *Jupyter notebooks* that make use of the *Pandas* and *Matplotlib* Python libraries.

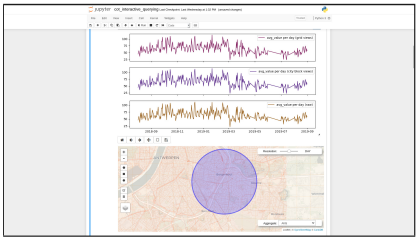
Serving Layer A REST API is implemented for serving client applications. In the PostgreSQL-based implementation (see Figure 3.10a), this API is provided by using the *Flask* web framework for Python and *NGINX+uWSGI* as an application server, while in the distributed stream processing approach (Figure 3.10b), this API runs on a *Jetty servlet container*. This REST API consists of two endpoints: one for handling *historical-spatial* queries and the other for *snapshot-temporal* queries. The specification of each of the API endpoints is presented below in table 3.1. Multiple instances of the API server are deployed to balance the load and to provide high availability.

Table 3.1: API specification for the serving layer (default values are shown in underlined text).

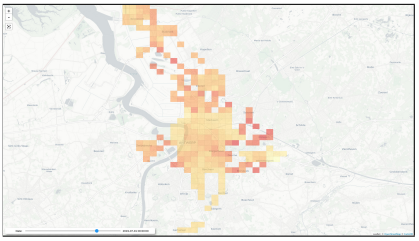
HS queries: `GET /airquality/{metric_id}/aggregate/{aggregate}/history`	
Path parameters	• `metric_id`: (*required*) one of the air quality metrics available from the Bel-Air setup (`no2`|`pm25`|`pm10`|`o3`|`...`).
	• `aggregate`: (*required*) one of the available aggregate function (`AVG`|`SUM`|`COUNT`).
Query paramenters	• `q_polygon`: (required) Well-Known Text (WKT) representation of the polygon selected by the user, e.g.: "`POLYGON ((1.0 0.0, 1.0 1.0, 0.0 0.0, 1.0 0.0))`".
	• `source`: tile grid (`tiles`), street blocks (`street_blocks`) or raw sensor data (`raw`).
	• `time_res`: `min`|`hour`|`day`|`month`.
	• `grid_precision`: in case multiple continuous views corresponding to multiple values of geohash precision have been computed, via this parameter it is possible to specify the desired precision for the query at hand (default: 6).
	• `from`: the start of the query interval as a timestamp in milliseconds.
	• `to`: the end of the query interval (exclusive) as a timestamp in milliseconds.
	• `interval`: optionally it is possible to use one of five predefined intervals: `5min`|`1hour`|`1day`|`1week`|`1month`.
ST queries: `GET /airquality/{metric_id}/aggregate/{aggregate}/snapshot`	
Path parameters	Same as for the previous endpoint
Query paramenters	• `bbox`: (*required*) comma-separated string of coordinates corresponding to the bounding box over which the snapshot would be taken.
	• `source, time_res, grid_precision`: same as for the previous endpoint.
	• `snap_ts`: timestamp in milliseconds corresponding to the instant the snapshot would be taken.

Client applications Two Jupyter notebooks are deployed as client applications, one implementing the first two use cases described in Section 3.4.2 and another implementing the third use case. Figure 3.11 shows screen captures taken from these implementations.

Figure 3.11 Screen captures of the Jupyter notebooks implementing the target use cases defined in Section 3.4.2.



(a) Use cases 3.4.2.1 and 3.4.2.2.



(b) Use case 3.4.2.3.

These notebooks consume the API available in the serving layer to resolve the *historical-spatial* and *snapshot-temporal* queries that support the interaction with end users.

3.4.3.1 Spatial Time-Series Database Approach

Processing on ingestion PostgreSQL triggers are used to implement the ingestion procedure described in Algorithm 3.1. These trigger functions are invoked for each of the sensor readings being consumed from the *Kafka broker*, relaying them to the corresponding continuous views for aggregation before being persisted into the time-series storage. Two spatial fragmentation schemas have been laid over the region covered by the mobile sensors, namely a *tile grid* built according to the *geohash* encoding algorithm by Niemeyer G. [48] (see Figure 3.1a for reference) and a grid corresponding to the *street-blocks* of the city of Antwerp (see Figure 3.1b). Additionally, four aggregation frequencies were considered, fragmenting time into minutely, hourly, daily, and monthly bins. In consequence, under this setup, eight continuous views (2-spatial fragmentation schemas $\times$ 4-aggregation frequencies) are computed, holding data summaries that comprise the results of three aggregate functions applied over the incoming stream of sensor observations: the arithmetic average of the measured values (`AVG`), the sum of the measurements (`SUM`), and number of reported readings (`COUNT`).

Storage and query processing layer For these layers, three open-source extensions of PostgreSQL are set up on top of this database engine, enabling it to store and query time-series data, to support geospatial operations, and to incrementally create and persist continuous views:

- *TimescaleDB* [49] is a time-series database working on top of PostgreSQL, thus being able to offer a *full SQL* querying interface while supporting fast data ingestion. Raw sensor readings consumed from the *Kafka broker* are formatted and stored into a TimescaleDB *Hypertable*, which partitions data in the temporal dimension for efficient ingestion and fast retrieval.

- *PostGIS* [50] is a spatial extension that allows PostgreSQL to store and query information about location and mapping. With PostGIS in place, the GeoJSON specifications of the *tile* and *street-block* grids are stored as two spatial tables, for which the records correspond to individual tile/street-block from the spatial fragmentation schemes. Likewise, each one of the records from the TimescaleDB Hypertable are augmented with a PostGIS geography object that corresponds to the sensor reading location. This enables the execution of spatial join operations required later during the querying stage to address calculations such as *point-in-polygon* and *polygon intersection*.

- *PipelineDB* [51] is an extension that enables the computation of continuous aggregates on time-series data, storing the results into regular PostgreSQL tables. The eight continuous views mentioned earlier are created in PipelineDB and incrementally computed as continuous queries running against the stream of sensor observations being handed in through the trigger functions in the ingestion layer. For illustration, listing 3.1 presents the SQL statement used in PipelineDB for creating a view that computes the three stated aggregates on a per-minute basis.

```
CREATE VIEW aq_no2_minutely_view WITH (action=materialize) AS
    SELECT fragment_id, observed_var, minute(time) AS ts,
        COUNT(*) AS count,
        SUM(value) AS sum_value,
        AVG(value) AS avg_value
    FROM aq_no2_stream -- stream of NO2 sensor measurements
    GROUP BY fragment_id, observed_var, ts;
```

Listing 3.1: Example of a view creation statement in PipelineDB.

Since PostgreSQL is the underlying storage technology used in this setup, it is possible to translate the procedures for handling *historical-*

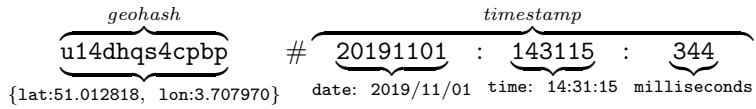
spatial and *snapshot-temporal* queries (from Algorithms 3.2 and 3.3, respectively) into declarative SQL statements, leveraging the expressiveness of this language along with the capabilities of the implemented extensions. An example of said statements is presented below in listing 3.2.

```
SELECT observed_var, ts, combine(avg_value) AS avg_value
FROM aq_no2_minutely_view
INNER JOIN tile_grid ON aq_no2_minutely_view.fragment_id =
    tile_grid.id
WHERE ST_Contains(ST_GeomFromText('<QUERY_POLYGON>'), tile_grid.
    geom)
GROUP BY observed_var, ts
ORDER BY ts; -- QUERY_POLYGON: Well-Known Text (WKT) representation
             -- of the user's polygonal selection.
```

Listing 3.2: Example of a HS query statement running on PipelineDB.

3.4.3.2 Distributed Stream Processing Approach

Processing on ingestion A *Kafka streams* application is implemented for this layer, according to the procedure in Algorithm 3.1. The Kafka streams library provides an API for conducting distributed stateful transformations on the feed of sensor observations being pushed to the *Kafka broker* by enabling multiple stream processor instances to consume the partitioned Kafka topics that the sensor readings are being written to. In consequence, the global application state is also partitioned into a distributed key-value store, instances of which are collocated with the working stream processors. Since Kafka streams does not support spatial operations out-of-the-box, in order to set up a statiotemporal fragmentation schema, a compound record key was associated to each of the incoming sensor observations, consisting of their *geohash* code (a base 32 sequence of 12 characters encoding the latitude and longitude of the measurement), along with their correponding timestamp, formatted as in the example shown below:



By augmenting sensor observations with keys structured in this way, the implemented ingestion procedure is able to set up a geohash-based spatial grid, leveraging the fact that readings sharing the first k-geohash characters fall into the same geospatial region identified by such k-character prefix. Likewise, the same procedure uses timestamp prefixes to set up a time-partitioning layout over the incoming stream

of sensor readings, pushing records into minutely, hourly, daily, and monthly bins. Thereafter, data summaries are continuously computed on each of the geohash-based spatial fragments for each of the time partitions, and their results are persisted into the distributed state store. As an illustration, listing 3.3 presents an example of the continuous views generated by the Kafka streams application.

```
...
u14dhq#20191101:140000:000: {AVG: 54.32, SUM: 182678.16, COUNT
    :3363},
u14dhq#20191101:150000:000: {AVG: 32.10, SUM: 111964.80, COUNT
    :3488},
u14dhq#20191101:160000:000: {AVG: 45.13, SUM: 147755.62, COUNT
    :3274},
u14dhq#20191101:170000:000: {AVG: 90.08, SUM: 304560.48, COUNT
    :3381},
...
```

Listing 3.3: Example of a continuous view with hourly time bins in *Kafka Streams*. The segment presented corresponds to the spatial fragment identified by the geohash prefix `u14dhq`.

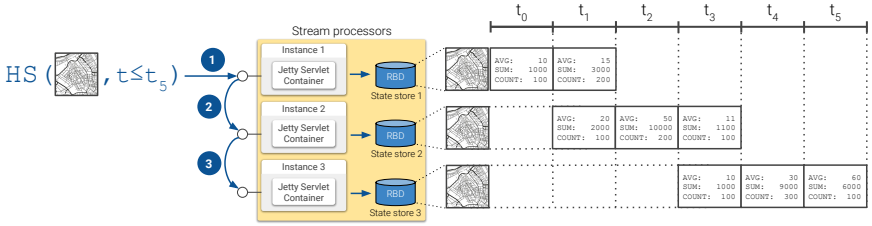
Storage layer This layer is also supported by tools provided by Kafka: raw sensor observations are stored into Kafka topics, while continuous views generated in the ingestion layer are stored into a distributed key-value database known as RocksDB [52], which Kafka uses as the default state store for stream applications. While records stored in Kafka topics are not directly queryable, continuous views in RocksDB allow simple key-based lookup and range queries. This is why a major part of the query processing needs to be conducted in the serving layer, when handling the client application requests.

Query processing and serving layer In this distributed setup, an instance of the REST API serving client requests is hosted on each of the Kafka stream processors. Each of these instances is only capable of answering queries on the portion of the application state available to the hosting stream processor. Therefore, resolving a query on the global state requires combining the results computed on the state available to each of the stream processor thus far. Consider for instance the example presented in Figure 3.12, illustrating the procedure for a setup with three stream processors, resolving a *historical-spatial* query: (1) The query reaches one of the instances of the serving layer API. This instance processes the query against the version of the continuous view persisted on its own state store. (2) Then, the query is relayed to a second instance to retrieve the data summaries from its corresponding state store and to combine them with those obtained from the

first instance. (3) This process is repeated until the query reaches the last API instance. Finally, the resulting sequence of aggregated data summaries is retrieved to the client application.

It is worth noting that the simplicity of the querying interface offered by the state stores—limited basically to key-based lookups and range queries—along with the key-value data model they adopt pay off in terms of query processing time, as will be shown when discussing the performance of these proof-of-concept implementations in the following section.

Figure 3.12 Procedure for distributed query resolution.



3.5 Experimental Evaluation

The previous section explored two proof-of-concept implementations of the EXPLORA framework, proving its ability to support typical use cases for visual exploratory applications on mobile sensor data. This section addresses a performance evaluation conducted on both implementations on a feed of air quality sensor observations collected from the *Bel-Air* smart city setup.

3.5.1 Query Accuracy Metric

The performance evaluation reported herein is mainly focused on determining to what extent the continuous computation of data summaries applied in EXPLORA effectively reduces the query response time on spatio-temporal data and what is the cost of such increase in responsiveness in terms of query accuracy. The latter was determined by defining a metric accounting for the average distance between the elements of the result sets obtained when querying continuous views—i.e., approximate answer—against those retrieved when querying the base raw sensor data—i.e., exact answer. Let $\boldsymbol{\mathcal{X}}_q$ and $\boldsymbol{\mathcal{Y}}_q$ be two result sets obtained from running a query q against both a continuous view $\boldsymbol{\mathcal{V}}$ and the base sensor data $\boldsymbol{\mathcal{R}}$, respectively. $\boldsymbol{\mathcal{X}}_q$ and $\boldsymbol{\mathcal{Y}}_q$ can be regarded as *relations* since they designate a set of ordered pairs:

$$\begin{aligned} \boldsymbol{\mathcal{X}}_q = & \{\langle k_{x1}, v_{x1}\rangle, \langle k_{x2}, v_{x2}\rangle, \ldots, \langle k_{x\mathsf{m}}, v_{x\mathsf{m}}\rangle\} \\ \boldsymbol{\mathcal{Y}}_q = & \{\langle k_{y1}, v_{y1}\rangle, \langle k_{y2}, v_{y2}\rangle, \ldots, \langle k_{y\mathsf{n}}, v_{y\mathsf{n}}\rangle\} \end{aligned} \tag{3.6}$$

With k_{xi} and k_{yi} being spatial-fragment identifiers or timestamps and v_{xi} and v_{yi} being aggregate values. Ideally, $\boldsymbol{\mathcal{X}}_q$ and $\boldsymbol{\mathcal{Y}}_q$ should have the exact same set of keys and values; this is, the distance between them (tuple-wise) should be zero. However, due to the applied spatio-temporal fragmentation scheme, data summaries—upon which queries are resolved—can only match spatial and temporal query predicates in an approximate manner. In consequence, key-value sets might differ from $\boldsymbol{\mathcal{X}}_q$ to $\boldsymbol{\mathcal{Y}}_q$. To estimate the average tuple-wise distance—henceforth, *distance*—between these two result sets, first, a *full outer-join* operation is computed:

$$\begin{aligned} \boldsymbol{\mathcal{Z}}_q &= \boldsymbol{\mathcal{X}}_q \rtimes\!\ltimes \boldsymbol{\mathcal{Y}}_q \\ &= \{\langle k_{z1}, (v_{xz1}, v_{yz1})\rangle, \langle k_{z2}, (v_{xz2}, v_{yz2})\rangle, \langle k_{z3}, (v_{xz3}, v_{yz3})\rangle, \ldots\} \\ & v_{xzi} = 0, \text{ if } k_{zi} \notin \boldsymbol{\mathcal{X}}_q \wedge v_{yzi} = 0, \text{ if } k_{zi} \notin \boldsymbol{\mathcal{Y}}_q \end{aligned} \tag{3.7}$$

Then, the distance (d) between these two result sets is estimated as follows:

$$\begin{aligned} d &: \boldsymbol{\mathcal{X}}_q \times \boldsymbol{\mathcal{Y}}_q \mapsto [0, 1], \\ d(\boldsymbol{\mathcal{X}}_q, \boldsymbol{\mathcal{Y}}_q) &= \frac{1}{|\boldsymbol{\mathcal{Z}}_q|} \sum_{i}^{|\boldsymbol{\mathcal{Z}}_q|} \frac{|v_{xzi} - v_{yzi}|}{|v_{xzi}| + |v_{yzi}|} \end{aligned} \tag{3.8}$$

where $|\boldsymbol{\mathcal{Z}}_q|$ denotes the cardinality of the set resulting from the outer-join operation in Equation (3.7). One appealing feature of this distance metric is that it provides a normalized symmetrical measure of the dissimilarity between two result sets, which makes it more easily interpretable than alternative distance metrics such as *dynamic time warping* (DTW) [53] used for measuring the similarity between two temporal sequences.

3.5.2 Experimental Setup

The data set collected for this performance evaluation covers about one-year's worth of sensor measurements (from August 2018 to August 2019) made to map the situation of air pollutant emissions over the city of Antwerp. The two proof-of-concept setups detailed in Section 3.4 were deployed to a Kubernetes cluster consisting of one master and three worker nodes, set up on the *imec/IDLab Virtual Wall* environment [54]. Table 3.2 lists the versions of the software tools used in these implementations.

Table 3.2: Versions of the software used in the experimental setup.

Software	**Version**
Kubectl	0.15.10
Linux Kernel	4.15.0-66-generic
Operating System	Ubuntu 18.04.3 LTS
Container Runtime Version	containerd://1.2.6
PostgreSQL (TimescaleDB + PostGIS + PipelineDB)	11.5 (1.4.2 + 2.5.2 + 1.0.0)
Apache Kafka	2.3.0
NGINX + uWSGI	1.14.2 + 2.0.17.1
Jetty Server	9.4.20.v20190813
Java (OpenJDK)	14-ea
Python	3.7.5

The process for collecting performance information on both proof-of-concept setups started by recording the queries generated during one user session on the first setup. This collection of queries—designated henceforth as *workload*—amounts to 222 statements comprising a wide range of query predicates (polygonal selections, timestamps, time intervals, etc.), 64% of which correspond to *historical-spatial* queries while the remaining 36% are *snapshot-temporal* requests. The collection of historical-spatial queries can be further divided into statements with a predicate in the temporal dimension (i.e., those querying over a certain period of time provided by the user) and queries without said predicate (namely, those querying over the whole period of available data thus far). Table 3.3 shows the final composition of the query workload, considering the discussed classification.

Table 3.3: Composition of the test workload used for the performance evaluation.

Query type	**# Queries**
HS (w/ temporal predicate)	90
HS (w/o temporal predicate)	52
ST	80
Total	**222**

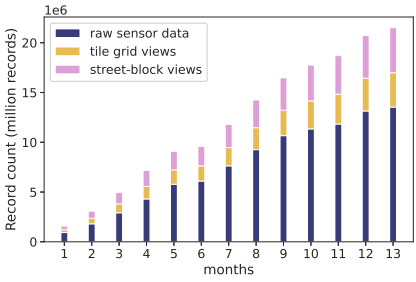
To determine how each of the EXPLORA implementations performs as the amount of ingested data increases, the test air quality data was fed to both setups in batches of one-month's worth of data. This way, at the end of each batch increment, the sequence of request included in the workload was run on both implementations while monitoring query response time and query accuracy. Each batch of raw sensor data was ingested and aggregated into a geohash-based tile grid (for which precision was set to a six-character geohash prefix) and—only for the spatial time-series database setup—a street-blocks based grid, which partitions the urban area of Antwerp into 12.230 polygonal regions.

3.5.3 Results

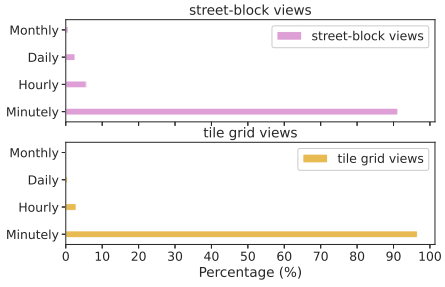
3.5.3.1 Continuous Views Storage Footprint

The continuous views generated through EXPLORA are by definition redundant data structures for read optimization [55] and, as such, entail a storage overhead. In this sense, Figure 3.13a illustrates the proportion of the number of records (i.e., data summaries) registered in the views w.r.t. the total count of raw sensor observations ingested per month for both tile and street-block grids. On average, tile grid views and street-block views amount, respectively, to 26.5% and 33.8% of the total record count for sensor readings. Since street-block views rely on a finer (and irregular) spatial fragmentation strategy than that used for tile views, the number of data summaries placed into the former views is larger in proportion to the amount of raw sensor observations.

Figure 3.13 Storage footprint of continuous views.



(a) Views grow proportionally to the ingested raw data.



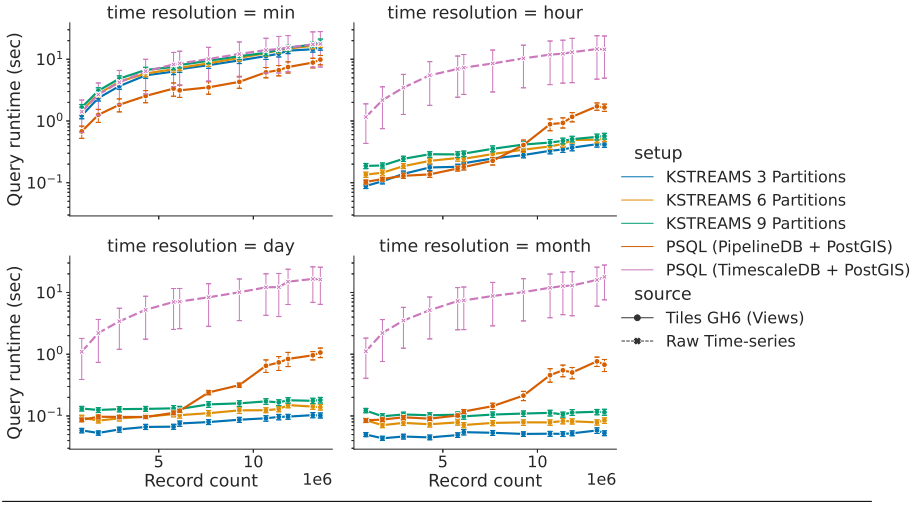
(b) *Minutely* views are proportionally the largest.

In the same vein, Figure 3.13b shows that most of the storage overhead is due to views with aggregation frequency set to one minute, accounting—in both tile and street-block views—for more than 90% of the total amount of generated data summaries. Again, as stated earlier in Section 3.3.2.2, the lower the resolution of the spatio-temporal fragmentation scheme, the smaller the size of the corresponding view: while for one-year's worth of sensor data, there might be around 8.640 *hourly* data summaries per spatial fragment, the corresponding *minutely* summaries would amount to 518.400, which explains the stark difference between the *minutely* view proportions and the second-largest *hourly* views.

3.5.3.2 Query Response Time for HS Queries without Time Predicate

When the serving API receives a *historical-spatial* request providing only the spatial parameter (and no conditions on the temporal dimension), the corresponding response is computed over the full extent of data available thus far. The response time reported for this kind of queries with regards to the amount of ingested sensor readings is illustrated in Figure 3.14. Results from multiple setups are presented in these charts in order to compare both implementations of EXPLORA. For the *spatial time-series database* approach (PostgreSQL based), the query response time on the raw sensor observations (*TimescaleDB + PostGIS*) and continuous views (*PipelineDB + PostGIS*) are reported; while for the *distributed stream processing approach* (Kafka based), results obtained from running Apache Kafka with three different partition settings are presented: 3 partitions/3 stream processors (*KSTREAMS 3 Partitions*), 6 partitions/6 stream processors (*KSTREAMS 6 Partitions*), and 9 partitions/9 stream processors (*KSTREAMS 9 Partitions*). Query processing time from the *TimescaleDB + PostGIS* setup serves as reference to estimate the performance gain in query response time for the remaining setups. These time measurements were conducted along the four considered temporal resolutions, namely *per-minute*, *per-hour*, *per-day*, and *per-month* bins. In light of the results obtained, it is worth highlighting four key facts:

Figure 3.14 Query response time vs. volume of ingested data: HS queries without time predicate.



(*i*) Query response time on the raw data (dashed line in Figure 3.14) behaves nearly the same along the four temporal resolutions, displaying a linear increase as the amount of data ingested grows larger. This describes an expected system's response, since each of these queries involves running expensive *sequential scan* operations over the full collection of raw sensor readings. This way, response time for these requests increases proportional to the amount of ingested sensor observations, regardless of the requested temporal resolution.

(*ii*) Continuous views (solid lines in Figure 3.14) in general outperform the base raw data for both implementations of EXPLORA. Only for views with per-minute temporal bins the performance benefit from using these synopsis structures is compromised due to the considerable size of said structures relative to the raw data (and to the remaining views, as evidenced earlier in Figure 3.13). However, even in this case, queries perform 1.1–1.3× faster in the *3-partition/3-processors* Kafka setup and 1.8–2.9× faster in the *PipelineDB + PostGIS* setup compared to queries running against the raw data. For the other considered time resolutions, queries running on the corresponding views perform up to two orders of magnitude faster than the reference setup, reaching sub-second response times in all cases.

(*iii*) When it comes to distributed stream processing, increasing parallelism—i.e., adding partitions and stream processors accordingly—actually leads to a slight decline in performance, which can be attributed to the overhead due to the process of combining the partial aggregates computed on each of the stream processors, which also implies data exchange among said processors (network overhead). That said, this approach still delivers a more stable response as the data volume grows compared to the spatial time-series approach, describing a linear-time trend for which the slope tends to zero as the temporal resolution of the aggregates decreases—notice the almost constant time for views with per-month temporal bins.

(*iv*) For ingested data under 6–8 million sensor observations, queries on the spatial time-series approach either outperform or closely follow the performance of those from distributed streaming setups. From 8 million records onwards, the query response time for the *PipelineDB + PostGIS* setup branches out, describing an exponential growth. In this situation, given the increased volume of data, indexed tables can no longer fit in the available memory; in consequence, parts of the index are repeatedly swap in and out of the database buffer pool, leading to a performance degradation.

3.5.3.3 Query Response Time for HS Queries with Time Predicate

This part of the evaluation deals with a more practical and sensible kind of query, namely those with predicates in both spatial and time dimensions. Figure 3.15 describes the performance for queries running on the six considered setups for five predefined time intervals: *last 5 minutes* and *last hour* running on *minutely* views; *last day* running on *hourly* views; and *last week* and *last month* running on *daily* views.

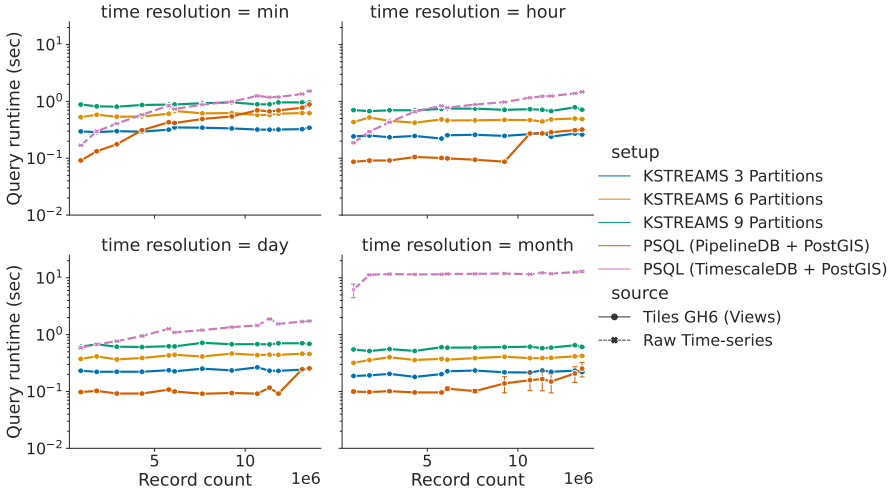
Figure 3.15 Query response time vs. volume of ingested data: HS queries with time predicate.

The obtained results show how the reference setup (*TimescaleDB + PostGIS*) is able to deliver almost constant-time performance for queries requesting *hourly* and *daily* time resolutions and outperforms the alternative implementations based on synopsis data structures with *minutely* time bins. This behaviour stems from TimescaleDB taking advantage of the inherent time-ordering of the ingested sensor observations to only process the most recent data. On the other hand, once again, the distributed stream processing approach stands out as the system with the most stable performance, featuring a nearly constant-time response as the amount of ingested data increases and sub-second query latency for all the considered time intervals. Meanwhile, the performance of the spatial time-series database (*PipelineBD + PostGIS*) approach falls behind, as it struggles to deliver a consistent time response as data grows larger.

3.5.3.4 Query Response Time for ST Queries

Snapshot-temporal queries provide a time-slice visual of the status of the observed variable over the geospatial region being displayed on the user's screen for a given timestamp and for a specific time resolution determining the span of time covered in the query computation (i.e., *one minute*, *one hour*, *one day*, or *one month*). Figure 3.16 below reports on the performance for this kind of query as the amount of data ingested increases.

Figure 3.16 Query response time vs. volume of ingested data: ST queries.



According to this test, the distributed stream processing setups deliver a constant-time response as data volume grows for all considered time resolutions in contrast to the alternative PostgreSQL setups, for which response is affected by the amount of data available (notice the linear-time performance for queries running with *one minute* time resolution) and the temporal interval over which the query is computed (notice how, for the reference setup, query latency tends to increase as this interval goes from *one minute* to one *one month*). This behaviour obeys to the fact that the distributed key-value database storing the partitioned continuous views enables constant-time key-based lookups, making the procedure implemented for resolving *snapshot-temporal* queries independent of the amount of data available and only subject to the size of the visible (or selected) geospatial region. Another significant result from this test is that, overall, both implementations of the EXPLORA framework deliver sub-second response times, proving these approaches effective to enable interactive-level performance for *snapshot-temporal* queries.

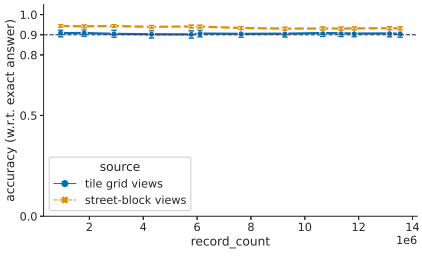
3.5.3.5 Query Accuracy on Continuous Views

The main caveat of using synopsis data structures for answering spatio-temporal queries is their inherent loss in accuracy. Figure 3.17 illustrates the level of accuracy attained in both of the proof-of-concept implementations of EXPLORA for different setups. In these charts, accuracy is defined as the complement of the distance metric formulated earlier in Section 3.5.1:

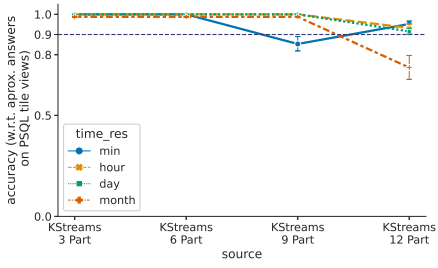
$$accuracy = 1 - d(\boldsymbol{\mathcal{X}}_q, \boldsymbol{\mathcal{Y}}_q)$$

where $\boldsymbol{\mathcal{X}}_q$ is a result set obtained upon running a query q on one of the available continuous views and $\boldsymbol{\mathcal{Y}}_q$ is a reference result set. For the spatial time-series database implementation, Figure 3.17a portrays the accuracy achieved for queries running on tile-based and street-block based views, as a function of the amount of data ingested. In this case, the reference result sets are those computed on the raw sensor observations for each query in the test workload. According to these results, the cost incurred in terms of accuracy is, on average, less than 10% for both types of views. It is also clear from the chart how using a finer spatial fragmentation schema allows for a more accurate approximation: queries running on the street-block views are 3.64 percent points closer to the exact answer than those running on the coarser tile-based views.

Figure 3.17 Query accuracy on the continuous views computed with EXPLORA: **(a)** Accuracy on both tile-grid and street-block views is above 90% on average. **(b)** Accuracy in relation to approximate answers from the *PipelineDB + PostGIS* setup: increasing the number of partitions eventually compromises query accuracy.



(a) *PipelineDB + PostGIS* setup.



(b) *Kafka streams* implementation.

On the other hand, for the distributed stream processing implementation, query accuracy is measured as a function of the number of partitions (and stream processors) used to split the ingested data and to generate the distributed continuous views under the premise that increasing parallelism im-

plies increased error probability. Since views from this setup are based on the same geohash tiles from the *PipelineDB + PostGIS* implementation, the query accuracy obtained in said implementation defines an upper bound for the distributed processing approach in this particular setup. That is why query accuracy in Figure 3.17b is estimated in relation to the approximate answers derived from the tile views of the PostgreSQL-based setup. The reported results indicate an effective drop in the expected accuracy once data and processing are split up into more that six partitions, evidencing that increased parallelism not only impacts query latency but also can eventually compromise the accuracy of the answers computed on continuous views.

3.6 Conclusions

Supporting visual-interactive exploration on top of the massive volumes of smart city data being generated nowadays remains largely an open problem. The stringent latency requirements typical of these kind of applications call for proactive and flexible data management mechanisms able to serve users with prompt answers to their information requirements, based on the most recent data available. In this sense, this chapter introduced EXPLORA, a microservice-based data management framework for spatio-temporal data produced in smart city environments (*i*) that leverages stream processing methods to continuously compute synopsis data structures over the live feed of measurements coming from mobile sensor, (*ii*) that defines a uniform interface to query said structures based on recurrent user interaction patterns, and (*iii*) that monitors system and query performance.

The experimental evaluation conducted on two proof-of-concept implementation of EXPLORA—one based on a traditional spatial time-series database approach and another using a distributed stream processing pipeline—proved the feasibility of the proposed framework, being able to serve expensive spatio-temporal queries with sub-second performance over a continuously increasing amount of sensor data (reaching up to 2 orders of magnitude speedup in comparison to queries running on the base raw observations) at the expense of less than 10% loss in accuracy and around 30% of storage overhead.

A current limitation of the EXPLORA framework is that the set of aggregate operations used for building the continuous synopsis structures (e.g., *average*, *sum*, and *count* in the described implementations) has to be defined upfront. In this sense, future work on this research will extend the framework to incorporate a *pluggable* mechanism that enables developers/users to provide custom aggregates as extensions that would be integrated to the running data ingestion pipeline. Additionally, the query processing com-

ponent of the framework will be further developed to enable features such as *predictive caching* to anticipate the queries that are likely to be issued next, according to user's interaction behaviour, and *federated querying* by implementing a *linked data fragments* interface, which boosts system scalability by pushing part of the query computation to the client-side application [56, 57].

Acknowledgements

This work was supported by the Research Foundation Flanders (FWO) under grant number G059615N—"Service oriented management of a virtualised future internet".

Addendum

***Further details on the experimental setup*:** The original manuscript this chapter is based upon lacks some relevant details regarding the sensor data sources used for the evaluation of the proposed approach. Table 3.4 below provides further information that complements the details provided in section 3.5.2.

Table 3.4: Information on sensor data sources

Parameter	**Value**
Number of sensors	78
Communication protocol	LoRaWAN
Sensed Measure	Nitrogen Dioxide (NO_2)
Average rate	27 readings/min
Total dataset size	13542770 readings

***Note on non-associative aggregates*:** The implementations of the EXPLORA framework detailed in this chapter only support *distributive/associative* and *algebraic* aggregate operations [45]. *Holistic* (non-associative) aggregates such as *median*, *rank*, among others are less trivial to implement in a distributed stream processing context due to their inherent dependency on global ordering of data. To incorporate this category of functions into the EXPLORA's ingestion pipeline, a solution based on probabilistic data structures such as *Count-min sketch* and *Hyperloglog* can be devised [58].

References

[1] Ruben Sánchez-Corcuera, Adrián Nuñez-Marcos, Jesus Sesma-Solance, Aritz Bilbao-Jayo, Rubén Mulero, Unai Zulaika, Gorka Azkune, and Aitor Almeida. Smart cities survey: Technologies, Application Domains and Challenges for the Cities of the Future. *International Journal of Distributed Sensor Networks*, 15(6):1550147719853984, 2019.

[2] Colin Harrison, Barbara Eckman, Rick Hamilton, Perry Hartswick, Jayant Kalagnanam, Jurij Paraszczak, and Peter Williams. Foundations for Smarter Cities. *IBM Journal of research and development*, 54 (4):1–16, 2010.

[3] Rodger Lea, Mike Blackstock, Nam Giang, and David Vogt. Smart cities: Engaging Users and Developers to Foster Innovation Ecosystems. In *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 1535–1542, 2015.

[4] Carina Veeckman and Shenja Van Der Graaf. The City as Living Laboratory: Empowering citizens with the Citadel toolkit. *Technology Innovation Management Review*, 5(3), 2015.

[5] Mila Gascó-Hernandez. Building a Smart City: Lessons from Barcelona. *Commun. ACM*, 61(4):50–57, mar 2018. ISSN 0001-0782. doi: 10.1145/3117800. URL https://doi.org/10.1145/3117800.

[6] Sumedha Chauhan, Neetima Agarwal, and Arpan Kumar Kar. Addressing Big Data Challenges in Smart Cities: a Systematic Literature Review. *info*, 2016.

[7] Bhagya Nathali Silva, Murad Khan, and Kijun Han. Towards Sustainable Smart Cities: A review of trends, architectures, components, and open challenges in Smart Cities. *Sustainable Cities and Society*, 38: 697–713, 2018.

[8] Ovidiu-Cristian Marcu, Alexandru Costan, Gabriel Antoniu, María Pérez-Hernández, Radu Tudoran, Stefano Bortoli, and Bogdan Nicolae. Storage and Ingestion Systems in Support of Stream Processing: A Survey. *[Technical Report]RT-0501, INRIA Rennes - Bretagne Atlantique and University of Rennes 1, France.*, 2018.

[9] K. Zoumpatianos and T. Palpanas. Data Series Management: Fulfilling the Need for Big Sequence Analytics. In *2018 IEEE 34th International*

Conference on Data Engineering (ICDE), pages 1677–1678, April 2018. doi: 10.1109/ICDE.2018.00211.

[10] Harish Doraiswamy, Eleni Tzirita Zacharatou, Fabio Miranda, Marcos Lage, Anastasia Ailamaki, Cláudio T. Silva, and Juliana Freire. Interactive Visual Exploration of Spatio-Temporal Urban Data Sets Using Urbane. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1693–1696, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450347037. doi: 10.1145/3183713.3193559. URL https://doi.org/10.1145/3183713.3193559.

[11] Chaowei Yang, Keith Clarke, Shashi Shekhar, and C. Vincent Tao. Big Spatiotemporal Data Analytics: a Research and Innovation Frontier. *International Journal of Geographical Information Science*, pages 1–14, 2019. doi: 10.1080/13658816.2019.1698743. URL https://doi.org/10.1080/13658816.2019.1698743.

[12] Jing He, Haonan Chen, Yijin Chen, Xinming Tang, and Yebin Zou. Diverse Visualization Techniques and Methods of Moving-Object-Trajectory Data: a Review. *ISPRS International Journal of Geo-Information*, 8(2):63, 2019.

[13] Raghu Ganti, Mudhakar Srivatsa, Dakshi Agrawal, Petros Zerfos, and Jorge Ortiz. MP-Trie: Fast Spatial Queries on Moving Objects. In *Proceedings of the Industrial Track of the 17th International Middleware Conference*, Middleware Industry '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450346641. doi: 10.1145/3007646.3007653. URL https://doi.org/10.1145/3007646.3007653.

[14] Dakshi Agrawal, Raghu Ganti, Jeff Jonas, and Mudhakar Srivatsa. STB: Space Time Boxes. *CCF Transactions on Pervasive Computing and Interaction*, 1(2):114–124, 2019.

[15] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331, 1990.

[16] Robert Alan Kempke and Anthony J McAuley. Ternary CAM memory architecture and methodology, Nov 1998. US Patent 5,841,874.

[17] Hoang Vo, Ablimit Aji, and Fusheng Wang. SATO: a Spatial Data Partitioning Framework for Scalable Query Processing. In *Proceedings*

of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 545–548, 2014.

[18] Ahmed M Aly, Ahmed R Mahmood, Mohamed S Hassan, Walid G Aref, Mourad Ouzzani, Hazem Elmeleegy, and Thamir Qadah. AQWA: Adaptive Query Workload Aware Partitioning of Big Spatial Data. *Proceedings of the VLDB Endowment*, 8(13):2062–2073, 2015.

[19] Mirjana Pavlovic, Darius Sidlauskas, Thomas Heinis, and Anastasia Ailamaki. QUASII: QUery-Aware Spatial Incremental Index. In *21st International Conference on Extending Database Technology (EDBT)*, 2018.

[20] Francisco García-García, Antonio Corral, Luis Iribarne, and Michael Vassilakopoulos. Voronoi-Diagram Based Partitioning for Distance Join Query Processing in Spatialhadoop. In *International Conference on Model and Data Engineering*, pages 251–267. Springer, 2018.

[21] Eleni Tzirita Zacharatou, Darius Šidlauskas, Farhan Tauheed, Thomas Heinis, and Anastasia Ailamaki. Efficient Bundled Spatial Range Queries. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '19, page 139–148, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369091. doi: 10.1145/3347146.3359077. URL https://doi.org/10.1145/3347146.3359077.

[22] Shaohua Wan, Yu Zhao, Tian Wang, Zonghua Gu, Qammer H Abbasi, and Kim-Kwang Raymond Choo. Multi-Dimensional Data Indexing and Range Query Processing via Voronoi Diagram for Internet of Things. *Future Generation Computer Systems*, 91:382–391, 2019.

[23] Nivan Ferreira, Marcos Lage, Harish Doraiswamy, Huy Vo, Luc Wilson, Heidi Werner, Muchan Park, and Cláudio Silva. Urbane: A 3D Framework to Support Data Driven Decision Making in Urban Development. In *2015 IEEE conference on visual analytics science and technology (VAST)*, pages 97–104. IEEE, 2015.

[24] Syed Monjur Murshed, Ayah Mohammad Al-Hyari, Jochen Wendel, and Louise Ansart. Design and Implementation of a 4D Web Application for Analytical Visualization of Smart City Applications. *ISPRS International Journal of Geo-Information*, 7(7):276, 2018.

[25] Cesium-Consortium. CesiumJS-Geospatial 3D Mapping and Virtual Globe Platform. *Available: https://cesium.com/cesiumjs/ (accessed February 3rd, 2020)*, 2020.

[26] Zhenlong Li, Qunying Huang, Yuqin Jiang, and Fei Hu. SOVAS: a Scalable Online Visual Analytic System for Big Climate Data Analysis. *International Journal of Geographical Information Science*, pages 1–22, 2019.

[27] Anil Ramakrishna, Yu-Han Chang, and Rajiv Maheswaran. An Interactive Web Based Spatio-Temporal Visualization System. In *Advances in Visual Computing*, pages 673–680, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-41939-3.

[28] Xuequan Zhang, Mingda Zhang, Liangcun Jiang, and Peng Yue. An Interactive 4D Spatio-Temporal Visualization System for Hydrometeorological Data in Natural Disasters. *International Journal of Digital Earth*, pages 1–21, 2019.

[29] Nan Cao, Chaoguang Lin, Qiuhan Zhu, Yu-Ru Lin, Xian Teng, and Xidao Wen. Voila: Visual Anomaly Detection and Monitoring with Streaming Spatiotemporal Data. *IEEE transactions on visualization and computer graphics*, 24(1):23–33, 2017.

[30] Ling-Jyh Chen, Yao-Hua Ho, Hsin-Hung Hsieh, Shih-Ting Huang, Hu-Cheng Lee, and Sachit Mahajan. ADF: An Anomaly Detection Framework for Large-Scale PM2. 5 Sensing Systems. *IEEE Internet of Things Journal*, 5(2):559–570, 2017.

[31] Ahmed M Shahat Osman. A Novel Big Data Analytics Framework for Smart Cities. *Future Generation Computer Systems*, 91:620–633, 2019.

[32] Claudio Badii, Elefelious G Belay, Pierfrancesco Bellini, Mino Marazzini, Marco Mesiti, Paolo Nesi, Gianni Pantaleo, Michela Paolucci, Stefano Valtolina, Mirco Soderi, et al. Snap4City: a Scalable IOT/IOE Platform for Developing Smart City Applications. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 2109–2116. IEEE, 2018.

[33] Claudio Badii, Pierfrancesco Bellini, Angelo Difino, Paolo Nesi, Gianni Pantaleo, and Michela Paolucci. MicroServices Suite for Smart City Applications. *Sensors*, 19(21):4798, 2019.

[34] A Node-Red. Visual tool for wiring the Internet-of-Things. *Available: http://nodered.org (accessed February 3rd, 2020)*, 2016.

[35] Arthur de M Del Esposte, Eduardo FZ Santana, Lucas Kanashiro, Fabio M Costa, Kelly R Braghetto, Nelson Lago, and Fabio Kon. Design and Evaluation of a Scalable Smart City Software Platform with Large-Scale Simulations. *Future Generation Computer Systems*, 93: 427–441, 2019.

[36] Fernando Freire Scattone and Kelly Rosa Braghetto. A Microservices Architecture for Distributed Complex Event Processing in Smart Cities. In *2018 IEEE 37th International Symposium on Reliable Distributed Systems Workshops (SRDSW)*, pages 6–9. IEEE, 2018.

[37] Unai Aguilera, Oscar Peña, Oscar Belmonte, and Diego López-de Ipiña. Citizen-Centric Data Services for Smarter Cities. *Future Generation Computer Systems*, 76:234–247, 2017.

[38] Natalia Andrienko, Gennady Andrienko, and Peter Gatalsky. Exploratory Spatio-Temporal Visualization: An Analytical Review. *Journal of Visual Languages & Computing*, 14(6):503–541, 2003.

[39] Robert E Roth, Arzu Çöltekin, Luciene Delazari, Homero Fonseca Filho, Amy Griffin, Andreas Hall, Jari Korpi, Ismini Lokka, André Mendonça, Kristien Ooms, et al. User Sudies in Cartography: Opportunities for Empirical Research on Interactive Maps and Visualizations. *International Journal of Cartography*, 3(sup1):61–89, 2017.

[40] Zhicheng Liu and Jeffrey Heer. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[41] Ling Liu and M Tamer Özsu. *Encyclopedia of Database Systems*, volume 6. Springer New York, NY, USA:, 2009.

[42] Inc Kubernetes. Kubernetes: Production-grade container orchestration. *Available: https://kubernetes.io/ (accessed March 3rd, 2020)*, 2020.

[43] Inc Red Hat. Red Hat Openshift. *Available: https://www.openshift.com/ (accessed March 3rd, 2020)*, 2020.

[44] Software Foundation Apache. Apache Mesos. *Available: http://mesos.apache.org/ (accessed March 3rd, 2020)*, 2020.

[45] Jiawei Han, Micheline Kamber, and Jian Pei. 4 - Data Warehousing and Online Analytical Processing. In Jiawei Han, Micheline Kamber, and Jian Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann

Series in Data Management Systems, pages 125 – 185. Morgan Kaufmann, Boston, third edition edition, 2012. ISBN 978-0-12-381479-1. doi: https://doi.org/10.1016/B978-0-12-381479-1.00004-6. URL http://www.sciencedirect.com/science/article/pii/B9780123814791000046.

[46] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester. City of things: An integrated and multi-technology testbed for IoT smart city experiments. In *2016 IEEE International Smart Cities Conference (ISC2)*, pages 1–8, Sep. 2016. doi: 10.1109/ISC2.2016.7580875.

[47] Software Foundation Apache. Apache Kafka. *Available: https://kafka.apache.org/ (accessed March 3rd, 2020)*, 2020.

[48] Gustavo Niemeyer. Geohashing. *Available: https://obelisk.ilabt.imec.be/api/v2/docs/documentation/concepts/geohash/ (accessed March 3rd, 2020)*, 2008.

[49] Inc. Timescale. TimescaleDB: An open source time-series SQL database optimized for fast ingest and complex queries, powered by PostgreSQL. *Available: https://www.timescale.com/products (accessed March 3rd, 2020)*, 2020.

[50] PostGIS. Spatial and Geographic objects for PostgreSQL. *Available: https://postgis.net/ (accessed March 3rd, 2020)*, 2020.

[51] D. Nelson and J. Ferguson. PipelineDB: High-performance Time-Series Aggregation for PostgreSQL. *Available: https://www.pipelinedb.com (accessed March 3rd, 2020)*, 2020.

[52] Open Source Facebook. RocksDB: A Persistent Key-value Store for Fast Storage Environments. *Available: https://rocksdb.org/ (accessed March 3rd, 2020)*, 2020.

[53] Omer Gold and Micha Sharir. Dynamic Time Warping and Geometric Edit Distance: Breaking the quadratic barrier. *ACM Transactions on Algorithms (TALG)*, 14(4):1–17, 2018.

[54] imec/IDLab. Virtual Wall: Perform Large Networking and Cloud Experiments. *Available: https://doc.ilabt.imec.be/ilabt/virtualwall/index.html (accessed March 11th, 2020)*, 2020.

[55] Leandro Ordonez-Ante, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert, and Filip De Turck. A Workload-Driven Approach for View Selection in Large Dimensional Datasets. *J Netw Syst Manage*, 2020. ISSN 1064-7570. doi: 10.1007/s10922-020-09526-z. URL https://doi.org/10.1007/s10922-020-09526-z.

[56] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-Scale Querying through Linked Data Fragments. In *LDOW*. Citeseer, 2014.

[57] Julian Andres Rojas Melendez, Gayane Sedrakyan, Pieter Colpaert, Miel Vander Sande, and Ruben Verborgh. Supporting sustainable publishing and consuming of live Linked Time Series Streams. In *European Semantic Web Conference*, pages 148–152. Springer, 2018.

[58] Daniel Ting. Streamed approximate counting of distinct elements: Beating optimal batch methods. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 442–451, 2014.

4

Explora-LD: a Linked Data Fragments approach for interactive querying on mobile sensor data in Smart Cities

Building upon the ideas introduced earlier in the Explora *framework, this chapter presents* Explora-LD*, an open source platform for scalable and cost-efficient publication of data summaries computed on live spatio-temporal data.* Explora-LD *thrives on achieving a more balanced trade-off between server-side and client-side query processing while still providing interactive response times. The proposed platform incorporates a simplified querying interface based on* Linked-Data Fragments *(LDF) which limits the complexity of the queries that the server is able to handle, and encourages response reuse across multiple client sessions. By adopting this simplified interface, the proposed platform intents to hand part of query computation over to the client-side application to favour system scalability. Since* Explora-LD *is based on the framework described in the preceding chapter, it is worth noting that some content overlap is expected concerning the definition of the data synopsis and spatio-temporal fragmentation mechanisms the platform relies on.*

⋆⋆⋆

L. Ordonez-Ante, S. Vermandere, B. Van de Vyvere, P. Colpaert, G. Van Seghbroeck, T. Wauters, B. Volckaert and F. De Turck

Submitted to the IEEE Internet of Things Journal, 2021

Abstract Smart Cities should aim to provide instantaneous access to their ever-growing collections of time series data, allowing citizens to query the data interactively. Publishing these time series as Linked Open Data is hindered by the difficulty to predict what type of questions are going to be asked. A new trade-off needs to be investigated allowing fast query answering while offering flexibility in tailoring the questions to end-user needs. We developed EXPLORA-LD, an open source platform running on a real-world Smart City deployment measuring a.o. air pollution. EXPLORA-LD summarizes spatio-temporal data on ingestion and republishes it with a Linked Data Fragments (LDF) approach. Measuring server load and query execution time, we found that, on average latency remain largely constant regardless of the number of queries being issued within a one-hour window, while CPU and memory usage scale linearly and logarithmically with the load, respectively. The evaluation also shows that EXPLORA-LD is able to deliver incremental answers to user queries under sub-second latencies, favoring application responsiveness. In this case, this incremental approach to query processing comes at the expense of major server load when compared to EXPLORA, a similar query engine implementing a blocking interface. However, we show that EXPLORA-LD servers are effectively freed of most of the query processing in the long run, thanks to the inherent cacheability of the LDF interface. This in turn translates into cache nodes consuming less than 50% the amount of memory required by the alternative blocking interface, and nearly constant CPU usage under increasing load.

4.1 Introduction

Smart City initiatives are currently gaining momentum as a response to the challenges posed by the sustained growth of the urban population around the world [1]. Governments, business and research institutions are determined to harness the full potential of information technologies, to bring forth a vision of cities where decision-making processes around urban development are driven by data, resources are optimally allocated across all the services provided by the city, and citizens are connected and actively involved in shaping public policies. Internet of Things (IoT) and particularly sensor technologies are instrumental in enabling this vision [2]. Via sensors, cities can collect a wide variety of data: the crowdedness in certain areas,

traffic hot-spots, concentrations of trash, parking availability, temperature and air pollution, etc. To leverage the potential this data offers, interactive applications need to be built that consume this data and provide it in some form to consumers, for example, through visualizations in graphs and charts. These visualizations can offer clear insights and help in defining local policies or simply informing citizens. To effectively support the low-latency requirements inherent to these interactive exploratory applications, Smart Cities should provide instantaneous access to the historical sensor data they collect, while offering a querying interface that is cost-efficient, scalable and sustainable over time [3]. Given the ever-growing volume and variety of data typical of these environments, meeting the stated requirements remains an open problem.

In the scope of this challenge we have devised EXPLORA-LD, an open source platform that combines a proactive approach to data ingestion for speeding up exploratory spatio-temporal queries, and a Linked Data Fragments (LDF) approach to data publication for providing a scalable and responsive interface to Smart City sensor data. EXPLORA-LD leverages distributed stream processing for continuously deriving synopsis data structures from the incoming feed of raw sensor measurements, this way capturing the spatial and temporal trends of the sensed variables. The platform we propose takes advantage of a spatial partitioning strategy, and the inherent order of time series data to arrange these summary structures into discretized fragments, representing the temporal state of the sensed variables over a specific spatial region. We have defined a Linked-Data compliant data model to represent said fragments, adopting widely-used standard vocabularies. The formulated model incorporates a number of hypermedia controls that enable clients to automatically explore the partitioned arrangement of data summaries, and discover the fragments that are relevant to answer a particular query. In this way, by running exploratory queries against the computed data summaries, instead of the raw historical data, EXPLORA-LD manages to deliver responsive query performance, at the expense of some accuracy. At the same time, the LDF querying interface makes for a scalable cost-efficient data access method, while the associated Linked-Data data model facilitates handling the wide variety of sensor data sources by mapping them into a generic representation.

To evaluate this approach, we have developed a proof-of-concept implementation of EXPLORA-LD and tested its performance on sensor data collected from a real Smart City environment deployed in the city of Antwerp, Belgium. In order to estimate the overall effect of publishing data summaries through an LDF interface, we benchmarked EXPLORA-LD against a similar system exposing data summaries via a traditional blocking API, which we

presented in previous work [4] (cf. Chapter 3). In this process we measured the performance of the two systems under increasing load, in terms of query response time and demand of computing resources. The outcomes of this evaluation reveal that, when running on uncached fragments, the LDF interface falls behind the performance of the blocking API. However, once the fragments are cached, EXPLORA-LD gets the upper hand both regarding CPU and memory utilization, and perceived responsiveness, since it is able to instantaneously deliver intermediate answers to the submitted queries.
The EXPLORA-LD platform provides a complete data processing pipeline for enabling interactive data exploration on streams of mobile sensor data. The approach we propose builds upon our previous work on the EXPLORA framework [4], which integrates support for ingesting raw sensor measurements and derive spatiotemporal-indexed data summaries out of live data streams. On top of this data processing pipeline, EXPLORA-LD offers a semantically-enriched data model and a novel LDF interface for publishing said summary data structures in a scalable, cost-efficient manner. It should be noted that the main contribution of the EXPLORA-LD approach lies in the definition of said domain model and lightweight LDF querying interface. Together these components provide a mechanism to expose live data summaries to client-side applications which works independently of the persistence model (i.e., how data is structured and laid out on disk). In this sense, even though the current implementation of EXPLORA-LD uses a Kafka-based ingestion pipeline and persistence backend [5], the domain model and querying interface the proposed platform incorporates are portable to streaming contexts supported in other data processing frameworks.
In this chapter we provide a detailed description of the EXPLORA-LD approach, including architectural aspects, implementation and evaluation. The following section addresses the related work. Section 4.3 discusses the mechanisms supporting the operation of the proposed platform. Section 4.4 elaborates on the devised data model, as well as the definition of the platform's architecture, while section 4.5 presents the experimental evaluation and results. Finally, section 4.6 examines the main findings of this research and provides pointers towards future work.

4.2 Related Work

4.2.1 Semantic Web technologies and Smart Cities

EXPLORA-LD leverages semantic Web technologies such as *Linked data* and shared vocabularies to foster interoperability and to enable automated

clients to navigate through Smart City data using hypermedia controls. In the same vein, the *IES Cities* platform, conceived by Aguilera et al. [6] harnesses linked open data along with crowd-sourced data to enable the creation of citizen-centered mobile services. This platform provides uniform access to Smart City data sourced in structured formats such as RDF, JSON, and CSV and relational databases through a RESTful interface, and implements a *query mapper* that enables users of the platform to query any registered data source using SQL statements. Similarly, Consoli et al. [7], propose a semantic data model aiming to ensure interoperability at the concept level for a number of heterogeneous data sources typical of Smart City contexts (sensor data, public transportation, waste collection, etc.). Data aligned with this model is published as RDF triples and made available through a dedicated SPARQL endpoint. As argued by Verborgh et al. [8, 9], SPARQL endpoints are expensive to host at high availability, since they enable clients to run arbitrarily complex queries. In contrast to the above approaches, EXPLORA-LD draws on the *Linked Data Fragments* (LDF) [8] conceptual framework, to come up with a lightweight access interface that efficiently offloads query processing from servers to clients, and favors cache reuse.

Jansen et al. [10] develop a conceptual study on *Big Open Linked Data* (BOLD) and how it allows for the creation of Smart City services and applications by linking and combining data sources, and by employing data analysis and predictive analytics. The authors highlight one of the main challenges in using BOLD for enabling Smart Cities stems in the data publishing mechanisms, however they do not elaborate on the issue any further. Along these lines, the European Telecommunications Standards Institute (ETSI) has released NGSI-LD [11], a standard specification that defines a Linked-Data compliant data model and an open API for publishing context information sourced from IoT devices. Sensors and other context sources are modeled as *Entities* according to the NGSI-LD data model. Each *Entity* is identified via a URI, and described in terms of *Properties* and *Relationships* to other *Entities*. The API allows clients to provide, consume and subscribe to context information, and enables near real-time access to the data encoded in the context entities. However, the NGSI-LD specification does not yet contemplate data aggregation capabilities.

On the other hand, two separate studies by Poveda-Villalon et al. [12] and Gyrard et al. [13] develop the case for the use of ontology catalogs for annotating *Internet of Things* (IoT) and Smart City data. The authors remark the importance of ontology catalogs since they encourage ontology reuse among application developers, hence promoting data integration. Most of the approaches in this category, including those in References [14–16], ac-

knowledge the convenience of semantic Web technologies to tackle the issue of interoperability among disparate Smart City data sources, but overlook the issue of providing efficient publishing methods and scalable access interfaces to data. Moreover, they lack querying mechanisms that enable interactive exploration over historical geolocated data typical of Smart City environments.

The EXPLORA-LD approach heavily relies on stream processing. Among the Semantic Web technologies there are solutions such as CQELS [17] or *Continuous SPARQL* (C-SPARQL) [18], commonly known as RDF Stream Processing (RSP) engines. While these engines by definition operate over streams of data encoded as RDF triples, the approach we propose is able to process streams of raw sensor readings. Additionally, these engines require maintaining open connections with each client querying the stream of RDF data, which poses a critical scalability bottleneck [19].

4.2.2 Interactive exploratory analysis of Smart City sensor data

Approaches in this category are mainly aimed at providing end-users with responsive client-side applications for conducting visual exploration over historical Smart City data. *4D CANVAS* by Murshed et al. [20] is a Web-based application that enables dynamic visualization of 3D geospatial data for supporting decision making in Smart Cities, leveraging a *WebGL*-based framework known as *Cesium* [21]. One downside of this application is that it requires the data to be residing on the file system (offline data). In a similar note, Ferreira et al. propose *Urbane* [22], a 3D visual framework tailored for architects and city planners to assist the decision making process around urban development projects. *Urbane* supports the exploration and visualization of both 2D and 3D data sets, being able to handle requests over several million observations with nearly sub-second performance. The *SOVAS* analytic system developed by Li et al. [23] supports interactive querying analytics over large climate data sets. *SOVAS* accesses climate data stored in the *Hadoop distributed file system* (HDFS) [24] and offers a SQL interface through distributed query engines such as Apache Hive [25] and Apache Impala [26]. A shared limitation of the approaches presented thus far in this category (and similar ones including [27, 28]) is that they are not able to handle live feeds of sensor readings, let alone conducting stream processing on said data.

Alternatively, approaches such as those conceived by Cao et al. [29] and Cheng et al. [30] feature online data processing mechanisms to support interactive applications for anomaly detection applications in Smart Cities.

Voila by Cao et al. is a visual interactive system able to process streams of traffic sensor data for identifying anomalous incidents. Moreover, the interaction mechanism implemented in *Voila* harnesses the users' judgment to better spot atypical events. On the other hand, the *ADF* framework proposed by Cheng et al. supports the detection of anomalous sensor readings, based on a statistical-based technique called *time-sliced anomaly detection* (TSAD). TSDA leverages metrics of spatial proximity between sensing devices to label unexpected readings. Both *Voila* and the *ADF* framework are tailored to the anomaly detection use case, while EXPLORA-LD aims at serving a more generic exploratory analysis on recent and historical sensor data. Additionally, by embracing semantic technologies and publishing linked data, EXPLORA-LD promotes the interoperability with third party systems at the data level, while providing a scalable and responsive querying interface.

4.3 EXPLORA-LD: Enabling mechanisms

The proposal addressed in this chapter aims at supporting interactive incremental query answering on current and historical mobile sensor data, while providing a scalable and responsive interface based on a *Linked Data Fragments* approach. The EXPLORA-LD platform thrives on three key enabling mechanisms: (*i*) *data synopsis*, (*ii*) *spatio-temporal fragmentation* and (*iii*) *Summary time series fragments*. This section discusses how our platform embraces these mechanisms to serve the stated purpose.

4.3.1 Data Synopsis

Queries on spatio-temporal data are typically formulated in terms of specific constraints over both time and space dimensions. For instance, a user might want to know what the situation of air pollution in his neighborhood looks like at the moment, and how it has been changing over time. This request can be placed in the form of a polygonal selection on an interactive map, along with a specification of the querying time period. As discussed in our previous work on the EXPLORA framework [4] (cf. Chapter 3, section 3.3.1), this kind of interaction pattern has been designated as *general exploratory task* by Andrienko et al. [31] on a survey on the topic of exploratory analysis of spatio-temporal data. To serve these exploratory tasks, EXPLORA-LD harnesses the same synopsis data structures defined in EXPLORA,which are

continuously computed over the incoming stream of spatio-temporal data. Said synopsis data structures are by definition drastically smaller than the base data collection they are derived from, allowing for much faster query execution.

4.3.2 Spatio-temporal fragmentation

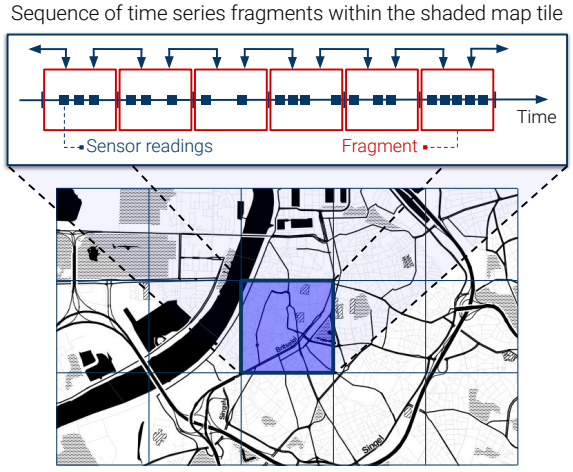
To build the synopsis data structures discussed in the previous section, EXPLORA-LD places a spatial fragmentation scheme over the incoming data stream, and computes a number of aggregates (e.g., *average*, *sum*, *count*) for several time resolutions (minutely, hourly and daily bins). We refer to the collection of aggregates under an individual spatial fragment and a certain temporal bin as *data summary*, and to the structured collection of data summaries as *continuous view*. At the time of writing the platform supports *Slippy Map* tiles as spatial fragmentation strategy, which is also used as geospatial indexing method in the *OpenStreetMaps* online service [32].
With this data ingestion pipeline in place, EXPLORA-LD is able to instantly serve queries supporting exploratory tasks, by publishing the incrementally computed data summaries as Linked Data Fragments, as we shall see in the section below.

4.3.3 Summary time series fragments

Linked Data Fragments (LDF) is a conceptual framework that encompasses all the existing interfaces to Linked Data datasets. According to the LDF framework interfaces are characterized in terms of how query processing cost is distributed between server and client, and what level of data granularity an interface is able to serve [8]. *Triple Pattern Fragments* are a type of Linked Data Fragments conceived as a low-cost method for publishing Knowledge Graphs at Web scale, by pushing part of the query processing to the client-side applications. With this method, the entries of a Knowledge Graph are arranged into fragments, each one matching a specific selector, together with metadata and hypermedia controls that enable clients to retrieve related fragments. Raw time series data can be transformed into times series fragments by applying the same approach. For time series fragments this fragmentation occurs in both the temporal and spatial dimensions: in the time dimension, the raw time series data gets fragmented into fixed-length time slots. Each time series fragment has a previous and a next attribute, containing URIs to navigate to the previous and the next fragment respectively along the temporal axis. In the spatial dimension, data is fragmented in tiles. These tiles are based on the Slippy Map tiles at a

certain zoom level. A visual representation of the fragmenting is shown on figure 4.1: each of the tiles in the map has a sequence of time series fragments associated, and each of the fragments contains timestamped sensor measurements that occurred within the bounds of that tile, ordered in time. The data structures described above, can also be referred to as *raw time series fragments*, since they deal with unprocessed sensor readings. *Summary time series fragments* (or *summary fragments*) on the other hand, follow a similar approach by arranging data summaries into fixed time slots and map tiles, instead of raw observations. Depending on its size (or length) a summary fragment may bundle one to several data summaries. For instance a one-day summary fragment might contain 1440 per-minute summaries, or 24 per-hour summaries, or one per-day summary.

Figure 4.1 Visual representation of the Time Series Fragments' structure.



Publishing data summaries as summary fragments changes the way in which clients interact with the server. Instead of sending a single request expecting one large response from the server (i.e. *blocking approach*), clients now send multiple requests to fulfill their query (i.e. *incremental approach*). Each request is resolved by transmitting a single summary fragment. The client then examines the next and previous attributes of the reply to find out which other fragments need to be requested to satisfy the original query. This form of interaction enables clients to answer queries in an incremental way, which makes for a more responsive user experience in visual exploratory applications. There are two main caveats to this approach: (*1*) queries involving arbitrary polygons are approximated to fit the tile grid. In that sense, summaries might include the contributions from readings outside the requested boundaries. (*2*) Since summary fragments might contain data

falling outside the requested time interval, it is up to the client to filter out said summaries when computing the final answer. While summary fragments can only guarantee an approximate answer to queries on spatio-temporal data, they are still able to provide relevant insights on the state and historical behaviour of the observed variables.

4.4 EXPLORA-LD: Data Model and Architecture

Based on the mechanisms discussed in the previous section, we have formulated a data model for publishing summary fragments, and designed and implemented a stream processing data pipeline for continuous computation of data summaries on live spatio-temporal sensor data. This section elaborates on both the model and the architecture of the proposed platform.

4.4.1 Data Model

The data model needs to facilitate the modeling of sensors and summaries of sensor observations and do this compliant to linked data standards. The *Semantic Sensor Network* Ontology (SSN) [33] was chosen, with its SOSA (*Sensor*, *Observation*, *Sample*, and *Actuator*) core ontology that contains the elementary classes and attributes. The data itself is represented in JSON-LD. Summary fragments are identified and accessible through URIs that match the following template:

```
data/{z}/{x}/{y}{?page,aggrMethod,
  aggrPeriod}
```

Where `z`, `x` and `y` identify the zoom level, and the x and y coordinates of the fragment tile, respectively; `page` is the `ISO` date of the day the aggregated sensor measurements were made; and `aggrMethod` and `aggrPeriod` define the requested aggregate method and the length aggregation interval (e.g. minute, hour, day), respectively. Each summary fragment features links to the previous and next fragment, along with a *Hydra* descriptor [34] enabling clients to automatically construct fragment URIs at runtime.
Likewise, a summary fragment is made up of aggregate observations. These aggregate observations are specified in terms of attributes such as the resulting aggregate, the start and end time of the aggregate interval, the list of devices reporting the measurements, and the observed variable. The listings below show an example of an aggregate observation and a summary fragment encoded in JSON-LD.

```
{
  "@context": {
    "sosa": "http://www.w3.org/ns/sosa/",
    "ssn": "http://www.w3.org/ns/ssn/",
    "time": "https://www.w3.org/TR/owl-time/",
  },
  "@id":"http://example.org/data/airquality.no2/1565110800000",
  "@type":"sosa:Observation",
  "sosa:hasSimpleResult":69.798,
  "sosa:resultTime":"2019-08-06T19:00:00.000Z",
  "sosa:phenomenonTime":{
    "time:hasBeginning":{
      "inXSDDateTimeStamp":"2019-08-06T19:00:00.000Z"
    },
    "time:hasEnd":{
      "time:inXSDDateTimeStamp":"2019-08-06T20:00:00.000Z"
    }
  },
  "sosa:observedProperty":"http://example.org/data/airquality.no2",
  "sosa:madeBySensor":[
    "http://example.org/data/lora.FF0032E81",
    "http://example.org/data/lora.FF0032EA2"
  ],
  "sosa:usedProcedure":{
    "@id": "http://example.org/data/id/avg",
    "@type": "ssn:Procedure",
    "hasOutput": {
      "@type": "ssn:Output",
      "count":268,
      "total":18705.991
    }
  },
  "sosa:hasFeatureOfInterest":"http://example.org/data/AirQuality",
}
```

Listing 4.1: Example of an aggregate observation

```
{
  "@id": "http://example.org/data/13/4196/2734?page=2019-08-06T00
      :00:00.000Z&aggrMethod=avg&aggrPeriod=hour",
  "@context": {
    "sosa": "http://www.w3.org/ns/sosa/",
    "ssn": "http://www.w3.org/ns/ssn/",
    "tiles": "https://w3id.org/tree#",
    "dcterms": "http://purl.org/dc/terms/",
    "hydra": "http://www.wr.org/ns/hydra/core#",
    "schema": "http://schema.org"
  },
  "tiles:zoom": 13,
  "tiles:longitudeTile": 4196,
  "tiles:latitudeTile": 2734,
  "schema:startDate": "2019-08-06T00:00:00.000Z",
  "schema:endDate": "2019-08-07T00:00:00.000Z",
  "hydra:previous": "http://example.org/data/13/4196/2734?page=2019-08-05
      T00:00:00.000Z&aggrMethod=avg&aggrPeriod=hour",
  "hydra:next": "http://example.org/data/13/4196/2734?page=2019-08-07T00
      :00:00.000Z&aggrMethod=avg&aggrPeriod=hour",
  "dcterms:isPartOf": {
    "@id": "http://example.org/data/13/4196/2734?aggrMethod=avg&
        aggrPeriod=hour",
    "@type": "tiles:Collection",
    "hydra:search": {
      "@type": "hydra:IriTemplate",
      "hydra:template": "http://example.org/data/{z}/{x}/{y}{?page,
          aggrMethod,aggrPeriod}",
```

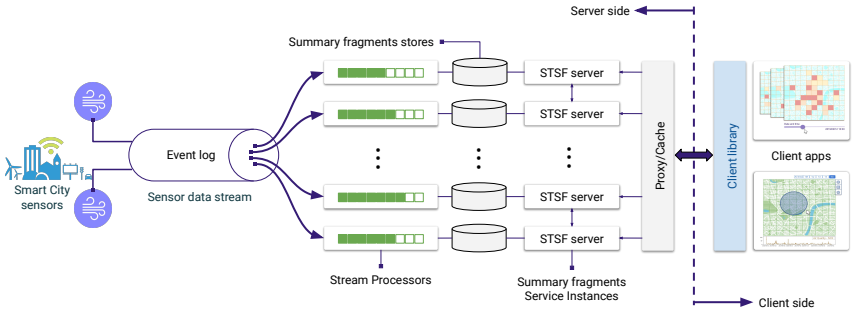
```
      "hydra:variableRepresentation": "hydra:BasicRepresentation",
      "hydra:mapping": [
        {
          "@type": "hydra:IriTemplateMapping",
          "hydra:variable": "x",
          "hydra:property": "tiles:longitudeTile",
          "hydra:required": true
        },
        {
          "@type": "hydra:IriTemplateMapping",
          "hydra:variable": "y",
          "hydra:property": "tiles:latitudeTile",
          "hydra:required": true
        },
        {
          "@type": "hydra:IriTemplateMapping",
          "hydra:variable": "z",
          "hydra:property": "tiles:zoom",
          "hydra:required": true
        },
        {
          "@type": "hydra:IriTemplateMapping",
          "hydra:variable": "page",
          "hydra:property": "tiles:timeQuery",
          "hydra:required": false
        },
        {
          "@type": "hydra:IriTemplateMapping",
          "hydra:variable": "aggrMethod",
          "hydra:property": "dcterms:accrualMethod",
          "hydra:required": false
        },
        {
          "@type": "hydra:IriTemplateMapping",
          "hydra:variable": "aggrPeriod",
          "hydra:property": "dcterms:accrualPeriodicity",
          "hydra:required": false
        }
      ]
    }
  },
  "@graph": [
    {/* aggregate observation 1 */},
    {/* aggregate observation 2 */},
    ...
  ]
}
```

Listing 4.2: Example of a Summary fragment

4.4.2 Architecture

We have designed EXPLORA-LD with scalability in mind. This is why we opted for a Microservices-based architecture. The platform consists of containerized, independently deployed components that can be elastically provisioned to meet the load requirements. For this we used Kubernetes [35] as container orchestration platform. Figure 4.2 shows the main components of the EXPLORA-LD platform, which contemplates two major parts: the server and the client side. The source code from both the

Figure 4.2 Architecture of the EXPLORA-LD platform.



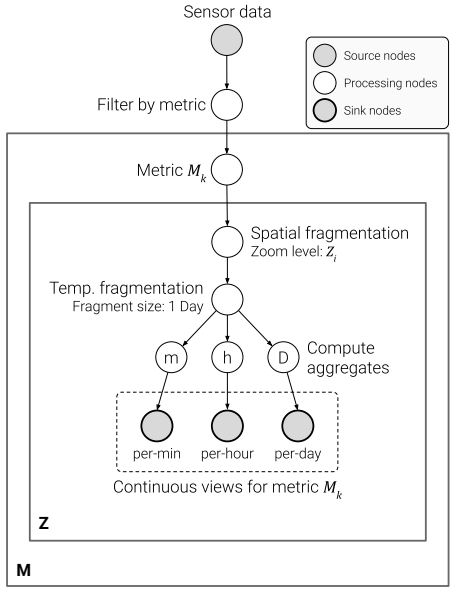
client and server sides of the implemented architecture is open source and available at https://github.com/IBCNServices/explora-ld (server side), and https://github.com/linkedtimeseries/timeseries-client (client library).

4.4.2.1 Server side

Event log This module serves as an interface between EXPLORA-LD and the sensor data providers. It collects the raw sensor measurements and relays the data stream to other components for further processing. The *event log* is implemented using *Apache Kafka* [5]. Kafka provides a suite of tools for processing streams of data, including a distributed message broker that adopts the *publish–subscribe* pattern. This way, incoming sensor observations are registered into a partitioned append-only log (namely *topic*), which maintains them over a fixed retention period. The Kafka broker allows multiple consumers to read and process the collected data in an asynchronous-concurrent way.

Stream processor This is one of the core modules of the platform's architecture. The stream processor subscribes to the Kafka topic where the raw sensor observations are being registered, and processes them to incrementally compute the data summary structures discussed in section 4.3. This component is implemented as a *Kafka streams* application, according to the data ingestion algorithm described in detail in one of our previous works [4]. Figure 4.3 presents a schematic of the implemented stream processing topology using *plate notation* [36] to represent the replication of certain regions of the graph. Notice that the stream processing pipeline defined this way supports multiple tile resolutions and multiple types of measurements: summary fragments can be computed over different zoom levels (given by the Z-index in

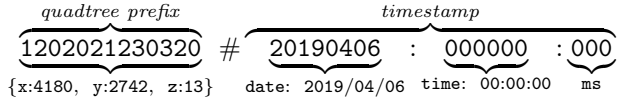
Figure 4.3 Representation of the stream processing topology using plate notation. Z stands for the different zoom levels used for spatial fragmentation, while M represents the collection of sensed variables (measures) available (e.g. *air pollution*, *temperature*, *humidity*, etc.)



the inner box of the topology), for several sensed variables (M-index in the outer box of the topology). Notice as well that continuous views of multiple temporal resolutions (per-minute, per-hour and per-day) are computed for each of the summary fragments, as indicated by the sink nodes in the topology. This enables the system to serve client requests with different spatial and temporal resolutions.

Summary fragments store The Kafka streams library provides an API that supports distributed stateful transformations on the stream of sensor readings, enabling multiple stream processor instances to consume the partitioned Kafka topics. Each one of these processor instances is responsible for maintaining a part of the global application state, namely the summary fragments corresponding to the sensor readings coming from each of the topic partitions. In the architecture diagram (Fig. 4.2) this is represented by the *Summary fragments stores*, which are distributed key-value database instances, co-located with the stream processors. Each summary fragment in these stores is associated with a compound record key. Said key consists of an identi-

fier of the summary fragment's tile encoded as its equivalent *quadtree prefix*, along with a timestamp indicating the start date of the summary fragment. As an illustration, consider the example shown below:



Each database instance manages a key range which is advertised to other modules via a *metadata service*. The *summary fragments server* (described next) uses this metadata service to determine which of the store instances should be queried in order to fetch a given summary fragment.

Summary time series fragments (STSF) server This component offers a HTTP interface to serve the summary fragments persisted in the key-value stores discussed above. Notice that there are multiple server instances, one per stream processor. Each instance is only able to serve the summaries from its local fragment store. Every client request for a (not-yet-cached) fragment reaches one of the available server instances, and it is up to the receiving server to decide whether to process the request, or forward it to another instance, based on the information provided by the metadata service of the summary fragment stores. This HTTP API runs on a *Jetty servlet container*.

Proxy/Cache This is the component where all HTTP requests arrive. In this cache, each requested fragment gets stored according to Least Recently Used policy. Notice that the historical data inside the fragments does not go stale and thus, are not removed from the cache after being unused for a certain amount of time. This is why `Cache-Headers` of all sent fragments, except the one of the current date, are set to $max - age = 31556952000$ (which corresponds to one year, the maximum value this directive is allowed to take according to the HTTP/1.1 specification). This does not apply for the fragment of the current date, since this one is still being updated with new measurements.

4.4.2.2 Client side

Client library As mentioned earlier, publishing data summaries as summary fragments involves a fundamental change in the way server and clients interact: instead of a traditional blocking approach, where clients send a single request per query and expect one large response

from the server, queries in EXPLORA-LD are handled through an incremental process in which clients request individual fragments one after the other using the hypermedia controls embedded into them, until they gather enough data to meet the query requirements. The *client library* is a proof-of-concept Node.js implementation that enable clients of EXPLORA-LD to do precisely this. The library takes a geographic area defined by a polygon, together with a time interval, an aggregate method (supporting *average*, *count*, and *sum* at the moment of writing), and an aggregate period (*min*, *hour*, or *day*). The query polygon is formed out of coordinate pairs and the time interval is defined as two ISO-date strings, for the start date and the end date of the query interval. With these inputs the following steps are performed:

1. The process starts by computing the minimum collection of Slippy map tiles to cover the provided geometry at the zoom levels supported by the platform. To illustrate this, consider the example in figure 4.4.

2. The following action is to start fetching the summary fragments for the requested time interval, beginning with the start date. For each subsequent date, the fragments for all tiles are requested.

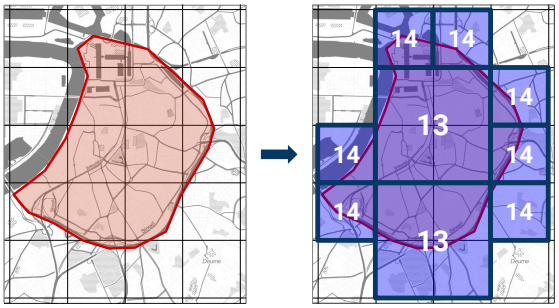
3. Once all data for a date has been gathered, all the summary fragments of the tiles are sorted into one object. This sorting is done according to the metric and `sosa:resultTime` attribute of the aggregate observations within the fragments.

4. Sorted aggregate observations are then combined to calculate the summary time series corresponding to the date being currently processed.

5. Finally, the procedure is repeated from step 2 onwards, until the end date of the requested time interval is reached.

All previous steps run asynchronous to the main thread. To notify users of the progress of their request, a listener system was created. In this way, *Client applications* can subscribe to certain events being triggered in the client library —for instance, whenever the library is done aggregating all the observations for a given date— and will be notified every time new data is available.

Figure 4.4 Computing the required tiles for the requested polygon: numbers within the tiles indicate the zoom level of each of them.



4.5 Experimental Evaluation

To estimate the benefits, costs and limitation of the proposed approach, we have conducted an empirical comparison of two setups: one implementing the stream processing pipeline and Linked Data Fragments interface proposed with the EXPLORA-LD platform, and the other featuring a similar distributed stream processing mechanism, but offering a traditional HTTP interface with a blocking query processing approach, which we refer to as EXPLORA [4]. More precisely, this evaluation uses the *Kafka Streams* implementation of EXPLORA detailed back in section 3.4.3.2 of the previous Chapter. This section describes first the experimental setup, and then discusses the results of the tests conducted.

4.5.1 Experimental Setup

The data used in this evaluation was sourced from the *Bel-Air* project: a Smart City setup that is currently operating in the City of Antwerp, under the *City of Things* (CoT) [37] initiative. This project is particularly concerned with collecting data on air quality in the city in a flexible, cost-efficient manner. Since the costs of deploying a maintaining a network of fixed sensors across the entire city could be prohibitively high, the *Bel-Air* project opted for a hybrid solution, involving mobile sensors installed on top of the mail delivery vans of the Belgian Postal service (*Bpost*), together with a number of sensors placed in fixed locations within the city.

From this setup we have collected about one-year's worth of sensor measurements (from August 2018 to August 2019) on a number of air pollutants over the city of Antwerp, which we fed into the EXPLORA-LD stream processing pipeline to compute summary time series fragments.

We have deployed both EXPLORA-LD and EXPLORA to a Kubernetes clus-

ter with one master and three worker nodes —all nodes sharing the same specifications: Intel E5645 @ 2.4GHz CPU, 8GB RAM, 50GB hard disk— using the infrastructure from the *imec/IDLab Virtual Wall* environment [38]. Both setups were spawned to run with five concurrent stream processors, each one subscribed to one of the five partitions of the Kafka topic to which the stream of sensor data was being posted. Then, we conducted a benchmark evaluation on both approaches measuring their performance in terms of query response time, CPU usage, and memory consumption. For this we used the query set defined by the polygons listed in table B.1 (See appendix B) —which cover the area of around 85 km^2 where sensor measurements were collected—, along with the time interval between May 3rd 2019, to June 3rd 2019. Each of these queries comprises multiple Slippy tiles in zoom levels 13 and 14. Table B.1 specifies the coordinates corresponding to each of the query polygons and the Slippy tiles they encompass. By drawing random combinations of these queries we were able to generate several workloads of various sizes, which we then used for the benchmark evaluation. The workloads we generated are available at https://github.com/LeandroOrdonez/explora-ld-test-workloads.
Both the EXPLORA-LD and EXPLORA setups were subjected to increasing load over a 1-hour time window, under the settings presented in table 4.1.

Table 4.1: Experiment settings.

Parameter	**Value**
Spatial fragmentation (tile zoom levels)	[13, 14]
Summary fragment size (EXPLORA-LD)	1 day
Aggregate period	1 minute
# queries/hour (workload size)	[4, 8, ..., 256, 512]

Considering the influence of caching on the query response time, we conducted these measurements for both cached and uncached summary fragments. Finally, the versions of the software tools used in these tests are listed in table 4.2.

4.5.2 Results

4.5.2.1 Query response time

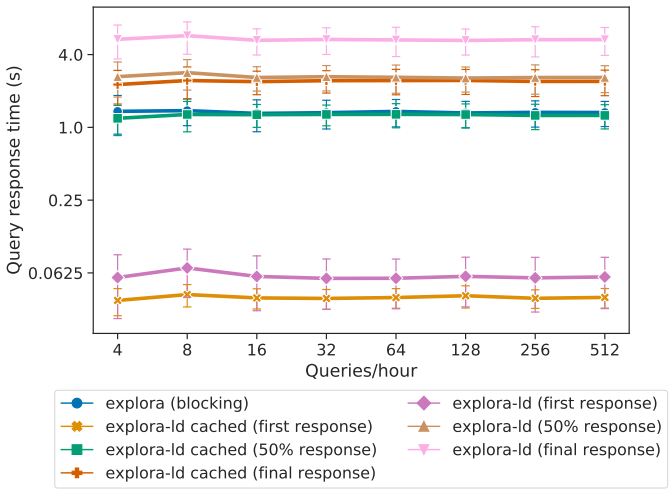
According to the procedure outlined earlier for the *client library*, the polygonal regions designated as the test queries get fitted by arrangements of four (4) Slippy tiles as shown in table B.1. Following said procedure, the client library translates each of these queries into 124 fragment requests (4 fragments $\times$ 31 days). We measured the time it takes for EXPLORA-LD to

Table 4.2: Versions of the software used in the experimental setup.

Software	**Version**
Kubectl	0.15.12
Linux Kernel (k8s pods)	4.15.0-101-generic
Operating System (k8s pods)	Debian GNU/Linux 10 (buster)
Container Runtime Version	containerd://1.2.6
Apache Kafka	2.3.0
NGINX (*proxy/cache* module)	1.19.0
Jetty Server	9.4.20.v20190813
Java (OpenJDK)	14-ea
Node.js	13.13.0

answer each fragment request under different load demands. Results of these measurements are presented in Figure 4.5, contrasted to those obtained for the alternative blocking approach (EXPLORA). For EXPLORA-LD —both cached and uncached fragments— average response times corresponding to the first, middle and last response are shown in the graph, while the time plotted for the blocking approach corresponds to the single response per query this interface yields by definition.

Figure 4.5 Query response time remains predominantly stable for all the evaluated setups under variable load.

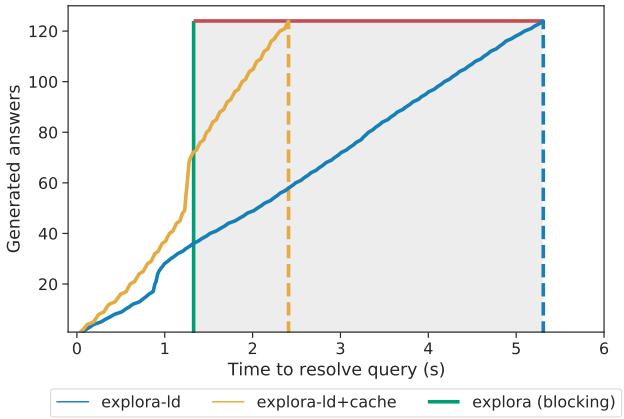


Notice that despite the increasing load, the average query response time remains largely invariant across all setups. EXPLORA-LD is able to provide the first answers under 62 milliseconds latency for both cached and

uncached fragments, which makes for a highly responsive interface. On the other hand, the EXPLORA (blocking interface) manages to serve a single (complete) answer in just under 1.5 seconds, around the same time it takes for EXPLORA-LD to reach half the number of answers to solve the query with cached fragments.

Figure 4.6 allows us to better visualize how the answers are incrementally delivered with EXPLORA-LD and how it compares to the behaviour of the alternative blocking interface. This graph displays the number of answers generated vs. the average query processing time. In the worst-case scenario —i.e. none of the requested summary fragments are cached— EXPLORA-LD takes on average 4x longer to provide a complete answer to the client's query, in contrast to the EXPLORA's blocking interface.

Figure 4.6 Response time performance: EXPLORA-LD delivered between 20% to 60% of the full query answer before the blocking interface computed a complete response. Nearly 30% of the answers were produced under one second, when querying cached fragments.



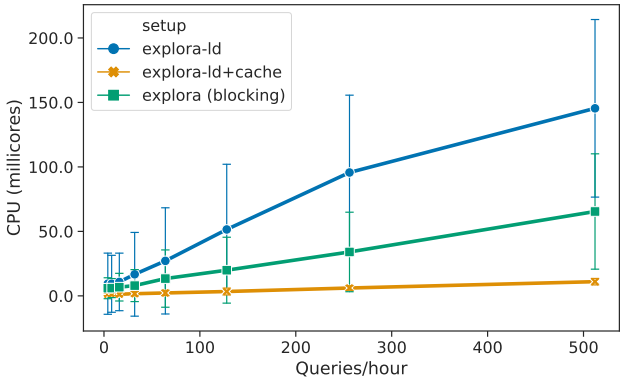
When instead all the requested summary fragments are cached (i.e. best-case scenario) EXPLORA-LD takes only 1.8x longer to completely resolve the query compared to the alternative approach. While this may be deemed as a significant latency, it is worth noting that EXPLORA-LD starts delivering partial answers nearly as soon as the query is submitted, in contrast to the blocking approach, which takes more than one second to produce a single, final answer. This is more noticeable for queries involving longer time intervals (for instance, several months). In those cases the blocking interface might take several seconds to obtain a final response, which can render client applications unresponsive.

4.5.2.2 CPU usage

We used the *Metrics API* provided by Kubernetes to collect measurements of CPU and memory consumption on the setups under evaluation, for both server and client sides. Figure 4.7 reports on the server's average CPU usage for the EXPLORA-LD implementation, compared to the alternative EXPLORA blocking interface. As illustrated in this graph, CPU utilization scales linearly with the amount of load in both setups. For EXPLORA-LD (with *uncached* fragments), this performance metric grows nearly 2.5x faster in comparison to the blocking interface, which is largely attributed to the overhead due to processing multiple request per query, instead of just one large request.

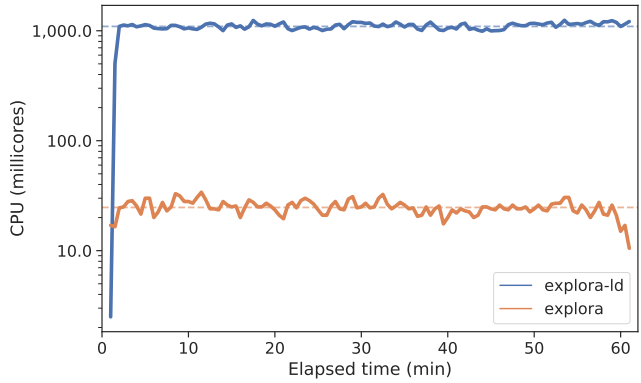
On the other hand, when requesting *cached* fragments, queries are served directly from the *proxy/cache* module, thus freeing server instances from performing any processing. In this sense, caching is one of the main advantages of the summary fragments approach we have adopted in EXPLORA-LD. In contrast to the blocking approach, the segmentation schema that this interface incorporates favors cache reuse across multiple clients, prompting an effective decrease in the amount of resources demanded from the server instances over time. Moreover, summary fragments can be cached indefinitely, since the data they are derived from —i.e. historic data— is not subject to change in the future.

Figure 4.7 Server side CPU Usage vs. queries/hour: CPU usage grows linearly with the amount of load for both EXPLORA-LD (with uncached fragments) and the blocking interface. When querying cached fragments the CPU usage of the NGINX *proxy/cache* instance is close to negligible.



As stated earlier in this chapter, the summary fragments interface that EXPLORA-LD implements thrives on pushing part of the query processing to the client-side applications. In this sense, it is expected for the clients of EXPLORA-LD to use more computing resources than those demanded by clients of a traditional blocking interface. Figure 4.8 compares clients of both setups in terms of their CPU usage throughout a session of 512 queries/hour. Under these high-load conditions, EXPLORA-LD clients use up to two orders of magnitude more CPU than clients from the blocking counterpart.

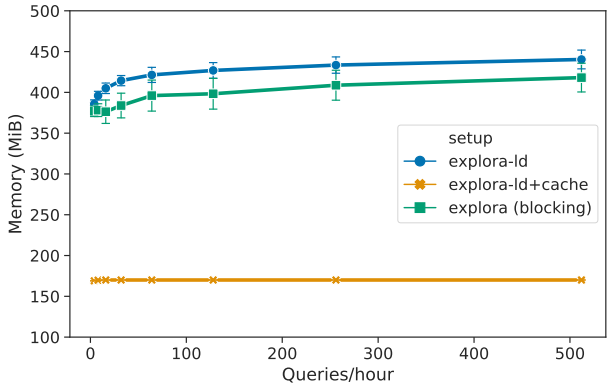
Figure 4.8 CPU usage on the client side under high load (512 queries/hour) is almost two orders of magnitude larger for EXPLORA-LD compared to the alternative EXPLORA blocking interface.



4.5.2.3 Memory Consumption

Memory consumption when requesting *uncached* fragments is also higher for EXPLORA-LD compared to the blocking interface. However, in contrast to CPU usage, the amount of memory demanded by the server instances grows nearly logarithmically as the load increases. As illustrated in the chart in Figure 4.9, server instances from the EXPLORA-LD setup consume around 6% more memory that servers of the blocking interface. The semantically-rich data model offered by the summary fragments interface demands more memory for producing query answers and serializing them as JSON-LD objects, in comparison to the plain tabular format served by the blocking interface. This explains the difference in memory requirements from both setups. Once the summary fragments are cached the amount of memory consumed by the *proxy/cache* module is remarkably small: around 2.3x lower than the EXPLORA's blocking interface, and 2.5x lower than EXPLORA-LD on uncached fragments.

Figure 4.9 Server side memory consumption: demand scales nearly logarithmically for Explora-LD on uncached fragments under increasing load while it remains largely constant for the same setup on cached fragments. This latter setup consumes 2.3x less memory than the alternative blocking interface.

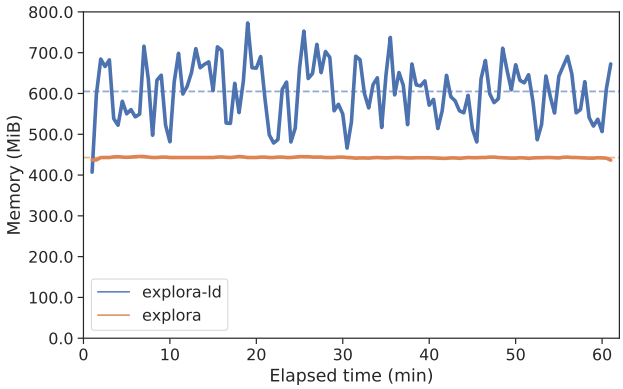


As for the client side, there is a noticeable difference between the memory used by EXPLORA-LD clients, and those from the EXPLORA blocking interface. Under medium-to-heavy load conditions, the procedure conducted on the *client library* (detailed back in section 4.4.2.2) to combine the retrieved summary fragments and compute the complete query answer, demands substantially more memory than that required for the traditional single-request/single-response approach. Figure 4.10 compares the distributions of measurements of memory consumption obtained on the client side, again across a session of 512 queries/hour. Under these conditions an EXPLORA-LD client requires on average nearly 1.4x the amount of memory consumed by a client of the blocking interface.

4.6 Discussion and Conclusions

As more Smart City initiatives are rolled-out, coming up with mechanisms that offer timely, efficient access to the streams of spatio-temporal data generated in these environments becomes critical for building the engagement among citizens and other stakeholders, required to ensure the sustainability of these initiatives over time. Aligned with these requirements, we conceived EXPLORA-LD, a data platform that enables the interactive exploration over historical and near-real time Smart City data. EXPLORA-LD harnesses the sequential nature of time series data and a quadtree-based spatial indexing structure known as *Slippy tiles*, to continuously generate relevant data sum-

Figure 4.10 Memory consumption on the client side under high-load conditions (512 queries/hour): a EXPLORA-LD client consumes —on average— as much as 1.4 times the memory used by a client from the blocking interface.



maries arranged into a graph of immutable connected resources, which then publishes through a Linked Data Fragments interface. This architectural choice involves a fundamental change in the way servers and clients interact: by offering a constrained interface, servers are freed from computing arbitrarily complex requests, and clients are compelled to conduct a major part of the query processing by traversing the linked structure of summary fragments. Another appealing feature of the EXPLORA-LD's fragments interface consists in its inherent cacheability: in general, once a fragment has been built as part of the answer to an arbitrary query, it can remain cached indefinitely for other clients to reuse it. As a consequence of this, recurrent queries are expected to run faster over time, while further decreasing the load on the server instances.

The data model conceived for representing the summary fragments provides a navigational structure via *Hydra*-compliant hypermedia controls. This comes with two significant benefits: (*i*) it enables automated clients to solve spatio-temporal queries by exploring the fragmented data themselves, and (*ii*) it promotes evolvability of the querying interface, in the sense that adding or removing features to the devised fragmentation will not cause existing clients to break, since they are able to dynamically interpret these changes *on-the-fly*. Additionally, by resorting to widely accepted vocabularies, the EXPLORA-LD data model provides metadata and formal semantics that facilitates the integration and interoperability with third party applications aligned with these Linked-Data vocabularies. In this chapter we

present a detailed definition of the EXPLORA-LD platform architecture, together with its Linked-Data compliant data model, and an implementation of the stream processing pipeline the platform thrives on. Additionally, we report on a performance evaluation of the proposed approach on real world air pollution data, collected from a Smart City setup running in the city of Antwerp. In this evaluation we compare EXPLORA-LD to a similar system exposing a traditional blocking query interface. In doing this, we contrast the response of these two systems to increasing load, keeping record of metrics such as query response time, CPU usage, and memory consumption. The results of this evaluation reveal a number of findings:

- Given a spatio-temporal query, EXPLORA-LD is able to provide a sequence of intermediate answers before the alternative EXPLORA blocking interface finishes processing and delivering a complete response. This is best perceived when performing exploratory requests over extended periods of time and/or large spatial regions. In those cases, the system offering a blocking interface could be dismissed as unresponsive due to the associated high latency, while EXPLORA-LD would promptly serve incremental answers regardless of the scope of the query.

- When running on uncached fragments EXPLORA-LD occupies more computing resources than the blocking interface. The foregoing is mainly attributed to the segmentation of queries into several requests, added to the *on-the-fly* serialization of the derived summary fragments into JSON-LD objects. However, once these fragments are cached, such resource-intensive serialization procedure is no longer required, leading to an effective decrease in the servers' load, and a reduction of around 50% in query response time.

- Queries on cached fragments are inexpensive in terms of CPU usage, and consume less than 50% the memory required by the EXPLORA blocking interface. By promoting cache reuse, the EXPLORA-LD approach makes for a cost-efficient scalable interface for supporting exploratory tasks on spatio-temporal data.

The proposed EXPLORA-LD platform thrives on a trade-off between query accuracy and query latency. In this sense, further experimentation is required to determine the optimal compromise between these two requirements. This would involve, for instance, testing the platform's performance on different spatial indexing mechanisms, and different fragment lengths.Likewise, optimizing the data serialization step —which proved to

be a rather expensive procedure— is going to be part of future iterations of our research. We are considering some alternative formats to JSON-LD, including *N-Triples*, *N-Quads*, and binary serialization formats such as *Avro*.

References

[1] Haimeng Liu, Chuanglin Fang, Yi Miao, Haitao Ma, Qiang Zhang, and Qiang Zhou. Spatio-temporal Evolution of Population and Urbanization in the Countries Along the Belt and Road 1950–2050. *Journal of Geographical Sciences*, 28(7):919–936, 2018.

[2] Ruben Sánchez-Corcuera, Adrián Nuñez-Marcos, Jesus Sesma-Solance, Aritz Bilbao-Jayo, Rubén Mulero, Unai Zulaika, Gorka Azkune, and Aitor Almeida. Smart cities survey: Technologies, Application Domains and Challenges for the Cities of the Future. *International Journal of Distributed Sensor Networks*, 15(6):1550147719853984, 2019.

[3] Bhagya Nathali Silva, Murad Khan, and Kijun Han. Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in Smart Cities. *Sustainable Cities and Society*, 38: 697–713, 2018.

[4] Leandro Ordonez-Ante, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert, and Filip De Turck. EXPLORA: Interactive Querying of Multidimensional Data in the Context of Smart Cities. *Sensors*, 20(9): 2737, 2020.

[5] Software Foundation Apache. Apache Kafka, 2020. URL https://kafka.apache.org/.

[6] Unai Aguilera, Oscar Peña, Oscar Belmonte, and Diego López de Ipiña. Citizen-centric data services for smarter cities. *Future Generation Computer Systems*, 76:234 – 247, 2017. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2016.10.031. URL http://www.sciencedirect.com/science/article/pii/S0167739X16304770.

[7] Sergio Consoli, Valentina Presutti, Diego Reforgiato Recupero, Andrea G. Nuzzolese, Silvio Peroni, Misael Mongiovi', and Aldo Gangemi. Producing Linked Data for Smart Cities: The Case of Catania. *Big Data Research*, 7:1 – 15, 2017. ISSN 2214-5796. doi: https://doi.org/10.1016/j.bdr.2016.10.001. URL http://www.sciencedirect.com/science/article/pii/S2214579616301289.

[8] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-Scale Querying through Linked Data Fragments. In Christian Bizer, Tom Heath, Sören Auer, and Tim Berners-Lee, editors, *Proceedings of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Work-*

shop Proceedings, April 2014. URL http://ceur-ws.org/Vol-1184/ldow2014_paper_04.pdf.

[9] Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. *Journal of Web Semantics*, 37–38:184–206, March 2016. ISSN 1570-8268. doi: 10.1016/j.websem.2016.03.003. URL https://linkeddatafragments.org/publications/jws2016.pdf.

[10] Marijn Janssen, Ricardo Matheus, and Anneke Zuiderwijk. Big and Open Linked Data (BOLD) to Create Smart Cities and Citizens: Insights from Smart Energy and Mobility Cases. In Efthimios Tambouris, Marijn Janssen, Hans Jochen Scholl, Maria A. Wimmer, Konstantinos Tarabanis, Mila Gascó, Bram Klievink, Ida Lindgren, and Peter Parycek, editors, *Electronic Government*, pages 79–90, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22479-4.

[11] ETSI NGSI-LD. Context Information Management (CIM) and Application Programming Interface (API). *ETSI GS CIM*, page V1.1.1, 2019.

[12] M. Poveda-Villalón, R. García-Castro, and A. Gómez-Pérez. Building an ontology catalogue for Smart Cities. In *eWork and eBusiness in Architecture, Engineering and Construction, Proceedings of the ECPPM 2014, 10th European Conference on Product & Process Modelling*, pages 829–839, Vienna, Austria, September 2014. CRC Press Leiden, Holanda.

[13] A. Gyrard, A. Zimmermann, and A. Sheth. Building IoT-Based Applications for Smart Cities: How Can Ontology Catalogs Help? *IEEE Internet of Things Journal*, 5(5):3978–3990, 2018.

[14] A. Gyrard and M. Serrano. A Unified Semantic Engine for Internet of Things and Smart Cities: From Sensor Data to End-Users Applications. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*, pages 718–725, 2015.

[15] Dan Puiu, Payam Barnaghi, Ralf Tönjes, Daniel Kümper, Muhammad Intizar Ali, Alessandra Mileo, Josiane Xavier Parreira, Marten Fischer, Sefki Kolozali, Nazli Farajidavar, et al. Citypulse: Large Scale Data Analytics Framework for Smart Cities. *IEEE Access*, 4:1086–1108, 2016.

[16] Ningyu Zhang, Huajun Chen, Xi Chen, and Jiaoyan Chen. Semantic framework of internet of things for smart cities: Case studies. *Sensors*, 16(9):1501, 2016.

[17] Danh Le Phuoc, Minh Dao-Tran, Anh Le Tuan, Manh Nguyen Duc, and Manfred Hauswirth. RDF Stream Processing with CQELS Framework for Real-Time Analysis. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, pages 285–292, 2015.

[18] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, EMANUELE DELLA VALLE, and Michael Grossniklaus. C-SPARQL: a continuous query language for RDF data streams. *International Journal of Semantic Computing*, 4(01):3–25, 2010.

[19] Ruben Taelman, Riccardo Tommasini, Joachim Van Herwegena, Miel Vander Sandea, Emanuele Della Valle, and Ruben Verborgh. On the Semantics of TPF-QS Towards Publishing and Querying RDF Streams at Web-scale. *Procedia Computer Science*, 137:43–54, 2018.

[20] Syed Monjur Murshed, Ayah Mohammad Al-Hyari, Jochen Wendel, and Louise Ansart. Design and implementation of a 4D web application for analytical visualization of smart city applications. *ISPRS International Journal of Geo-Information*, 7(7):276, 2018.

[21] Cesium-Consortium. CesiumJS-Geospatial 3D Mapping and Virtual Globe Platform, 2020. URL https://cesium.com/cesiumjs/.

[22] Nivan Ferreira, Marcos Lage, Harish Doraiswamy, Huy Vo, Luc Wilson, Heidi Werner, Muchan Park, and Cláudio Silva. Urbane: A 3D Framework to Support Data Driven Decision Making in Urban Development. In *2015 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 97–104. IEEE, 2015.

[23] Zhenlong Li, Qunying Huang, Yuqin Jiang, and Fei Hu. SOVAS: A Scalable Online Visual Analytic System for Big Climate Data Analysis. *International Journal of Geographical Information Science*, pages 1–22, 2019.

[24] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The HADOOP Distributed File System. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. IEEE, 2010.

[25] Jesús Camacho-Rodríguez, Ashutosh Chauhan, Alan Gates, Eugene Koifman, Owen O'Malley, Vineet Garg, Zoltan Haindrich, Sergey Shelukhin, Prasanth Jayachandran, Siddharth Seth, Deepak Jaiswal, Slim Bouguerra, Nishant Bangarwa, Sankar Hariappan, Anishek Agarwal, Jason Dere, Daniel Dai, Thejas Nair, Nita Dembla, Gopal Vijayaraghavan, and Günther Hagleitner. Apache Hive: From MapReduce to Enterprise-Grade Big Data Warehousing. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1773–1786, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450356435. doi: 10.1145/3299869.3314045. URL https://doi.org/10.1145/3299869.3314045.

[26] Marcel Kornacker and Alex Behm. *Impala*, pages 1–7. Springer International Publishing, Cham, 2018. ISBN 978-3-319-63962-8. doi: 10.1007/978-3-319-63962-8_253-1. URL https://doi.org/10.1007/978-3-319-63962-8_253-1.

[27] Anil Ramakrishna, Yu-Han Chang, and Rajiv Maheswaran. An Interactive Web Based Spatio-Temporal Visualization System. In *Advances in Visual Computing*, pages 673–680, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-41939-3.

[28] Xuequan Zhang, Mingda Zhang, Liangcun Jiang, and Peng Yue. An interactive 4D spatio-temporal visualization system for hydrometeorological data in natural disasters. *International Journal of Digital Earth*, pages 1–21, 2019.

[29] Nan Cao, Chaoguang Lin, Qiuhan Zhu, Yu-Ru Lin, Xian Teng, and Xidao Wen. Voila: Visual Anomaly Detection and Monitoring with Streaming Spatiotemporal Data. *IEEE transactions on visualization and computer graphics*, 24(1):23–33, 2017.

[30] Ling-Jyh Chen, Yao-Hua Ho, Hsin-Hung Hsieh, Shih-Ting Huang, Hu-Cheng Lee, and Sachit Mahajan. ADF: An Anomaly Detection Framework for Large-Scale PM2. 5 Sensing Systems. *IEEE Internet of Things Journal*, 5(2):559–570, 2017.

[31] Natalia Andrienko, Gennady Andrienko, and Peter Gatalsky. Exploratory Spatio-Temporal Visualization: an Analytical Review. *Journal of Visual Languages & Computing*, 14(6):503–541, 2003.

[32] Open Street Map. Slippy Map Tilenames, 2020. URL https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames.

[33] World Wide Web Consortium (W3C). Semantic Sensor Network Ontology – W3C Recommendation, 2017. URL https://www.w3.org/TR/vocab-ssn/.

[34] Markus Lanthaler. Hydra Core Vocabulary: A Vocabulary for Hypermedia-Driven Web APIs – Unofficial draft., 2020. URL https://www.hydra-cg.com/spec/latest/core/.

[35] Kubernetes Inc. Kubernetes: Production-grade container orchestration, 2020. URL https://kubernetes.io/.

[36] Eric Mjolsness. Labeled Graph Notations for Graphical Models: Extended Report. Technical report, Technical report, University of California Irvine–Department of Computer Science, School of Information and Computer Science, 2004.

[37] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester. City of Things: An Integrated and Multi-technology Testbed for IoT Smart City Experiments. In *2016 IEEE International Smart Cities Conference (ISC2)*, pages 1–8, Sep. 2016. doi: 10.1109/ISC2.2016.7580875.

[38] imec/IDLab. Virtual Wall: Perform large networking and cloud experiments., 2020. URL https://doc.ilabt.imec.be/ilabt/virtualwall/index.html.

5

EXPLORA-VR: Content Prefetching for Tile-based Immersive Video Streaming Applications

Chapter 4 introduced EXPLORA-LD *as a platform capable of serving interactive query responses, even under high-load conditions. This chapter goes further to study the challenge of adapting to variable data retrieval patterns in latency-sensitive applications while serving multiple concurrent clients. Using tile-based immersive video streaming services as a relevant use case, this chapter presents* EXPLORA-VR*, a network-assisted query preprocessing mechanism inspired by the framework discussed in chapter 3. In tile-based immersive video streaming, the user's quality of experience (QoE) is heavily affected by network latency. By analyzing the stream of requests issued by users with active streaming sessions,* EXPLORA-VR *intends to anticipate viewer actions (head movements), and prefetch the video content they are most likely to watch in the upcoming segments into a nearby cache-enabled edge server. Clients consuming video tiles served by these nearby edge nodes experience lower latency, higher network throughput, and in consequence, higher video quality and a smoother playback experience in comparison to conventional content delivery approaches.*

⋆⋆⋆

L. Ordonez-Ante, J. van der Hooft, T. Wauters, G. Van Seghbroeck, B. Volckaert and F. De Turck

Accepted for publication in the Journal of Network and Systems Management, 2022

Abstract Despite the growing popularity of immersive video applications during the last few years, the stringent low latency requirements of this kind of services remain a major challenge for the existing network infrastructure. Edge-assisted solutions compensate for network latency by relying on cache-enabled edge servers to bring frequently accessed video content closer to the client. However, these approaches often require historical request traces from previous watching sessions or adopt passive caching strategies subject to the *cold-start* problem and prone to playout freezes. This chapter introduces EXPLORA-VR, a novel edge-assisted content prefetching method for tile-based 360° video streaming. This method leverages the client's rate adaptation heuristic to preemptively retrieve the content that the viewer will most likely watch in the upcoming segments, and loads it into a nearby edge server. At the same time, EXPLORA-VR incrementally builds a dynamic collective buffer for serving the requests from active streaming sessions based on the estimated popularity of video tiles per segment. An evaluation of the proposed method was conducted on head movement traces collected from 48 unique users while watching three different 360° videos. Results show that EXPLORA-VR is able to serve over 98% of the client requests from the cache-enabled edge server, leading to an average increase of 2.5× and 1.4× in the client's perceived throughput, compared to a conventional client-server setup and a *least recently used* caching policy, respectively. This enables EXPLORA-VR to serve higher quality video content while providing a freeze-free playback experience and effectively reducing network traffic to the content server.

5.1 Introduction

The recent outbreak of the COVID-19 pandemic has forced a radical shift in reality for a vast majority of the human population. Given the strict restrictions on mobility and social contact, people were compelled to move several aspects of their daily life into the digital world. These circumstances have boosted the interest in 360° immersive video applications (*augmented and virtual reality—AR/VR*) as a means to provide realistic and engaging user experiences,that make up for the lack of presence and physical interac-

tion [1–3]. However, the stringent demands in terms of bandwidth and very low latency of AR/VR applications still represent a major challenge for the existing network infrastructure [4].

For services relying on VR headsets for content delivery, the delay perceived by the user is a critical factor for determining the overall experience. Research on this topic signals that the motion-to-photon (MTP) latency for VR displays should be less than 20 ms to prevent the perception of scene instability and *cybersickness* [5, 6]. For on-demand tile-based 360° video streaming in particular, many of the existing studies have focused on mitigating the effect of latency by increasing viewport prediction accuracy and applying *HTTP adaptive streaming* (HAS) methods to adapt the quality of the requested content to the network conditions [7, 8]. While these approaches achieve a rational use of the bandwidth as perceived at the client's side, the network latency due to distant content servers can still substantially degrade the viewer experience.

As an answer to this problem, network-supported solutions leveraging cache-enabled edge servers have been proposed [8, 9]. The idea behind these approaches consists of bringing frequently accessed video tiles closer to the client; this offsets the network delay, which in turn leads to a significant improvement in the quality of the delivered content. This is, however, easier said than done: the high variety of possible viewport configurations—due to the freedom of device orientation, added to different network conditions—makes it hard to determine a priori the set of tiles that should be cached. In this sense, network-supported solutions often rely on log traces obtained from previous streaming sessions to estimate the popularity of the content, and/or adopt passive caching strategies in which only those tiles that are requested get cached at the edge server. These approaches entail two fundamental problems: (*i*) historical request traces are not always available for every piece of content, and (*ii*) the *cold-start* problem: early users would barely experience any improvement from having a cache nearby, due to the fact that most of their requests for content end up being forwarded to the origin server.

To address these issues, in this chapter we introduce EXPLORA-VR, a content prefetching mechanism for tile-based immersive video streaming. Our solution introduces two fundamental changes in the traditional workflow of content consumption for this kind of services: (*1*) the early advertising of the outcome of the viewport prediction and rate adaptation algorithm, running on the *head-mounted display* (HMD), and (*2*) the incremental building of a collective buffer that incorporates fixation patterns shared by the viewers. The rationale behind this is two-fold:

1. The information on the predicted user's viewport is forwarded to the cache-enabled edge server before the client's device starts buffering content. The edge server uses this information to preemptively retrieve—at a given quality level—the video tiles that the user is likely to watch in the upcoming segments, and it loads them into memory. Then the client starts consuming the video content from a closer server. In these circumstances, a HAS client would perceive that the content is downloaded with low latency, leading to a high network throughput estimation, and consequently to an increase in quality of the requested video tiles.

2. Since having multiple clients consume the same VR content within a small time window is a common use case (e.g., the *on-demand*, *near-live* scenario when a content provider premieres a new release), we have devised a stream-processing pipeline which enables combining the different user predicted viewports into a dynamic collective buffer (henceforth referred to as *DCoB*) which is built and refined incrementally as new users join the streaming session. The purpose of this DCoB is to serve as a cache holding frequently accessed content, preventing the edge server from flooding the content server with duplicate requests.

This chapter presents the following three main contributions of the solution we propose: (*1*) an edge-assisted, content-agnostic mechanism that proactively downloads the video tiles that an individual viewer is likely to watch in the near future (*2*) the formal definition of the data structure and stream processing pipeline behind the DCoB, which enables low-latency delivery of 360° video content to multiple users taking part of an on-demand, near-live streaming scenario, and (*3*) the experimental evaluation of the proposed approach on a public dataset which comprises the viewport traces from 48 users, collected throughout immersive video sessions. We have benchmarked the EXPLORA-VR prefetching mechanism against a conventional client-server configuration (without caching/prefetching), and a setup implementing a traditional *least-recently used* (LRU) caching replacement policy. Results show that the devised prefetching mechanism substantially improves the quality of experience (QoE) perceived by the viewer, in terms of video quality, startup latency, and occurrence of playout freezes, while reducing the backhaul traffic and content server's load.
It should be pointed out that it is not in the scope of this chapter to reach an optimal trade-off between network resource consumption and video delivered quality, as is the case for approaches in the literature such as [10] and [11]. Our work is focused on investigating data processing methods for

enabling preemptive retrieval of immersive video content which are able to adapt to the fixation patterns of multiple concurrent viewers. To achieve this, we leverage the computing resources of cache-enabled edge nodes, and we rely on existing methods for client-side viewport prediction and tile-based rate adaptation such as those introduced in [12] and [13].
The remainder of this chapter is structured as follows. Section 5.2 discusses the related work. Section 5.3 describes the detailed description of the techniques behind the content prefetching mechanism for tile-based 360° video streaming. Section 5.4 elaborates on the architecture of a proof-of-concept implementation of the proposed approach. Section 5.5 presents the experimental setup and the results derived from the evaluation. Finally, conclusions and perspectives for further research are provided in Section 5.6.

5.2 Related Work

Immersive video applications are typically bandwidth-hungry and highly sensitive to latency. A large body of research in this field has been devoted to develop efficient mechanisms of content delivery. Existing approaches can be grouped into three categories according to the main focus of their respective contribution, namely *client-driven*, *server optimization*, and *edge-assisted* solutions.

5.2.1 Client-driven HAS streaming for tile-based 360° video

To improve transmission efficiency, approaches in this category divide an equirectangular projection of the spherical video into several rectangular areas of the same size, referred to as tiles. By implementing said tiling scheme, the client can opt to prioritize the tiles that overlap with the viewer's viewport and request them in a higher quality representation than the tiles that are not visible to the user. Representatives approaches of these tile-based viewport-dependent adaptive video streaming solutions include those by Hosseini [14], Xie et al. [15], Graf et al. [16], Nguyen et al. [17], and van der Hooft et al. [13]. These works are fundamentally focused on addressing two main challenges: (*i*) *viewport prediction*: anticipate user movements to ensure content is timely displayed following the *field of view* (FoV) of the user; and (*ii*) quality of experience (QoE): providing a smooth, responsive viewing experience at the highest possible video quality that the best-effort network can deliver [8]. In essence, these approaches adopt traditional HTTP adaptive streaming techniques, and augment them to support tile-based content delivery, while meeting the stringent demands in terms

of latency and interactivity of omnidirectional video streaming. Although these solutions allow for an efficient use of the link capacity, they are still highly sensitive to network latency due to content servers situated in distant locations which severely degrades the user experience.

5.2.2 Server optimization solutions

This category comprises works mainly focused on maximizing viewer's QoE while optimally allocating server and network resources. Long et al. [10] propose a solution to the problem of optimal transmission resource allocation on the server side given a specific requirement of video quality from the viewer, as well as the optimal encoding rate for each video tile given a certain transmission energy budget. The solution contemplates exploiting several multicast opportunities that involve balancing trade-offs between video quality, computation, and consumption of communication resources. One of the implications of the proposed multicasting mechanisms is that the server might transmit video tiles at a higher quality representation than requested by a certain client. In such a case, the client application would incur a processing cost in order to scale down the received video tile to the appropriate quality representation. Building upon [10], the work by Zhao et al. [11] investigates the impact of viewport prediction on adaptive streaming of tiled 360° video in a multi-carrier wireless system. The authors consider a setup with a multi-antenna base station from which video content is transmitted to one or multiple single-antenna clients. Within the scope of said setup, the authors propose a framework that optimizes the downlink subcarrier allocations as well as the encoding rates for tiles and FoVs at the server side. The solution proposed in [11] aims at maximizing the video quality delivered to the clients, while controlling the rebuffering time for different levels of certainty about the outcome of the viewport prediction. It is noteworthy that the optimization investigated in [11] relies on methods that operate on the radio link layer, which is out of the scope of the work we present in this chapter.

Another approach that fits within this category is introduced by Shi et al. [5], who propose a remote rendering solution in which the server is able to stream only the scenes within the user's FoV plus a margin area around it whose width depends on the perceived system latency. Instead of a tiling scheme, the server uses an adaptive cropping filter that adjusts the delivered content to the fraction of the VR video overlapping with the current user viewport. A design decision made by the authors consists of minimizing the use of video buffering to reduce the system's response latency. As a consequence, the proposed remote rendering solution is sensitive to network

jitter and prone to frame dropping. Furthermore, the authors do not provide a clear indication concerning the performance of the proposed solution under high server load (i.e., serving multiple concurrent viewers).

5.2.3 Edge-assisted solutions

Thanks to the recent availability of public datasets on Virtual Reality (VR) video streaming —such as those by Lo et al. [18], David et al. [19], Fremerey et al. [20], and Wu et al. [21], among others— there has been an increasing interest in investigating methods for mining behavioral patterns from user movement traces. According to the study by Rossy et al. [22], navigation trajectories followed by viewers with high affinity exhibit patterns that can be used for optimizing the content delivery in streaming systems. Approaches aligned with this idea are often labelled as *edge-assisted* or *network-supported* solutions. Papaioannou et al. [23] addressed the problem of optimal caching for tile-based VR video streaming in the wireless edge network. Specifically, the solution introduced in [23] formulates a tile and tile resolution caching policy that aims at minimizing the error between the cached and requested content. The authors studied a static caching scenario in which the caching decision is made upfront, based on statistical data of the tile resolution demands from past watching sessions. Similarly, Mahzari et al. [24] explored the application of edge caching as a measure to compensate for network latency, and offload the content servers and backhaul network. The authors of this work conceived a FoV-aware caching policy based on a bayesian model which takes in the sequence of requests made by previous viewers. The proposed model gauges the popularity of individual tiles, and makes decisions on which content to cache/evict based on said metric. Similarly, Maniotis and Thomos [25] devised a cache replacement strategy for tile-based omnidirectional video, supported by a *deep reinforcement learning* (DRL) framework. This strategy takes into account the popularity of both videos and individual tiles. The authors introduced the concept of *virtual viewports* defined as the most popular video tiles resulting from the overlapping FoV of multiple users. To learn the optimal policy for tile placement in the cache, the DRL framework first requires to train a deep neural network (DNN) on past user requests.

These approaches (and related proposals such as [26–29]) have proven the pertinence and substantial benefit of edge caching to improve QoE in 360° video services, while reducing the load on the core network. However, these solutions often require an *offline stage* in which they fit a certain data model to traces of user requests. Afterward, in a subsequent *online stage*, this model is used to make decisions on which content to cache/evict, according to the demands from new users consuming the streaming service. In

addition, the studies discussed above adopt a passive approach to caching, i.e. tiles are stored into the edge-server memory only after they have been requested. Under these circumstances, early viewers would experience little benefit from the caching strategy in place, an issue that is commonly referred to in the literature as the *cold-start* problem [30]. Using a cold cache translates into cache miss events, which in turn increases the likelihood of playback freezes, since user requests have to be relayed back to the content server, thus incurring additional latency. To counter this issue, we propose a new FoV-aware content prefetching approach for tile-based adaptive 360° video streaming. This approach takes advantage of existing viewport prediction techniques to preemptively retrieve and cache the video content that the viewer is most likely to consume in the upcoming segments. Additionally, this mechanism does not rely on training data from historical traces as it is able to learn a *collective viewport* on the fly, out of the requests made by viewers with active streaming sessions. The content inside the collective viewport dynamically adapts in response to the content that is most demanded by the audience at a given point in time, which makes this approach specially appealing for near-live immersive video streaming applications.

5.3 EXPLORA-VR: Approach Overview

Figure 5.1 High-level component view of the VR content prefetching scenario. The link between *Content* and *Prefetch* servers features a larger capacity and higher latency than the one between *Prefetch* server and *Client*.

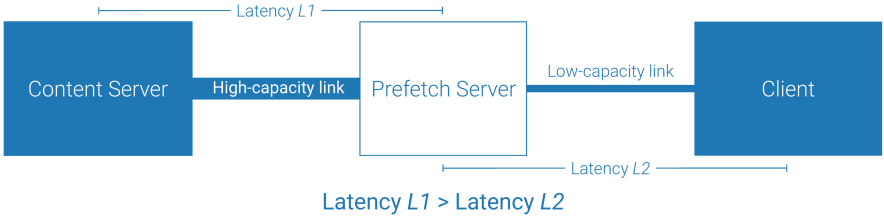


Figure 5.1 illustrates the components that make up the content prefetching mechanism we propose. This mechanism is deployed on a cache-enabled edge server acting as a transparent proxy between the client and the content server. In this section we elaborate on the techniques that lay the foundation of our solution, namely (*1*) the early advertising of the outcome of the viewport prediction and rate adaptation algorithm, and (*2*) the *dynamic collective buffer* (DCoB).

5.3.1 Viewport Prediction Advertising and Prefetching

In immersive video applications based on 360° video, it is common for content to be segmented not only in time but also in the spatial dimension. The HEVC/H.265 standard, for instance, allows to split an equirectangular projection of the content into $m \times n$ tiles of the same resolution. By adding this spatial dimension, clients can prioritize the content within the user's *field of view*, assigning a higher quality to specific regions of the video, hence making more optimal use of the bandwidth resources [31, 32].

To prevent buffer starvation and ensure a smooth playback in these highly interactive applications, traditional HAS methods need to be augmented. HAS clients for VR applications rely on techniques for predicting the users' target field of view or viewport, and rate adaptation heuristics to fine-tune the quality level of the requested content in response to the users' movements and network conditions [7, 16, 32].

Several methods have been introduced for viewport prediction in tile-based VR video streaming over the last years. On the one hand, *content-agnostic* approaches estimate the trajectory the viewer is likely to follow based on the viewport center locations of the last few milliseconds. To do so, some of these approaches use linear projection on the previous viewport positions [13, 31, 33, 34], while others rely on machine learning models trained on user movement traces [35, 36]. *Content-aware* techniques on the other hand, attempt to anticipate user movements based not only on an estimation of the viewer's trajectory, but also on specific features derived from the video content itself such as *image saliency*, *fixation density* and *object motion maps* [37–40].

In this work, we adopt the content-agnostic method proposed by van der Hooft et al. in [12] for predicting the user's viewport. In contrast to other content-agnostic solutions that assume the user moving on a path in the two-dimensional space defined by the equirectangular projection of the video, the method proposed in [12] models the viewer's movement as a trajectory on the *unit sphere*'s surface. In this way, the future location of the viewport center is estimated by unidirectionally extending the path covered by the viewer thus far across the surface of the unit sphere (*spherical walk*). This approach to viewport prediction provides a more natural approximation of the viewer's motion within the 360° video scene. This allows for a more accurate prediction compared to alternative content-agnostic solutions using linear extrapolation of the user's trajectory over the equirectangular projection of the video.

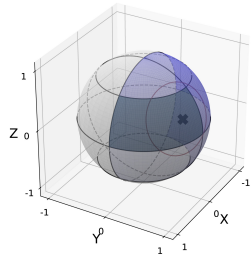
It is worth noting that the content prefetching mechanism we propose does not involve any substantial modification to the adopted viewport prediction scheme. Besides, while we favor the use of spherical-walk based viewport

prediction—mainly due to its enhanced accuracy—the devised prefetching mechanism is easily compatible with other alternative content-agnostic viewport prediction methods such as those proposed by Petrangeli et al. [31] and Xu et al. [34].

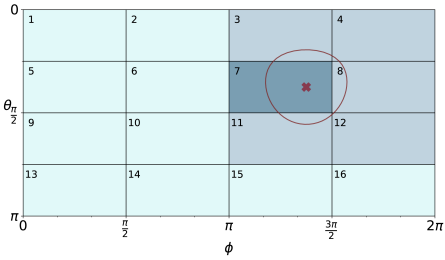
Along with the viewport prediction scheme based on spherical walks, we also adopt the *Center Tile First* (CTF) rate adaptation heuristic proposed in [13]. The intent behind this heuristic is to maximize the quality level for the video tiles located closer to the viewport center. In doing so, tiles from an equirectangular VR video are ranked according to the great-circle distance between their center and the viewport predicted location. The closer a certain tile is to the viewport center, the higher its priority and the quality representation that gets assigned to it.

As illustration, consider the example in Figure 5.2 for a 4×4 tiling scheme and two quality levels. The diagram outlines both the viewport (circular area on the sphere in Figure 5.2a) and viewport center (indicated as a cross mark). In this example, the CTF heuristic has prioritized the six tiles that lie closer to the viewport center, assigning them a high quality representation.

Figure 5.2 Example of the application of the CTF rate adaptation heuristic for a setup with a 4×4 tiling scheme and two quality levels: The highest quality representation gets assigned to the blue-shaded tiles, while the remaining ones are requested in the lowest quality. The number of high/low quality tiles in this example is arbitrary as it depends on the network conditions between client and server.



(a) Viewport projection in the unit sphere. Based on [32]



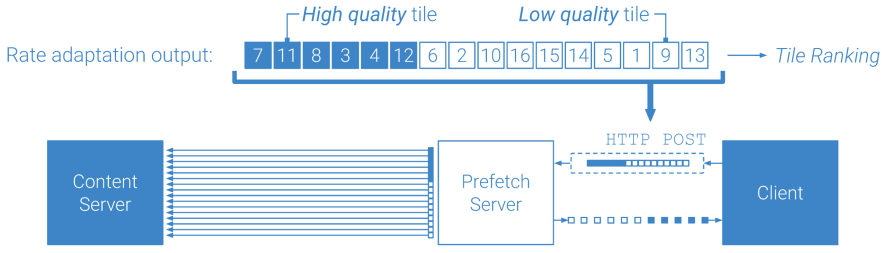
(b) Equivalent equirectangular projection of the viewport. Based on [32]

The output of the rate adaptation heuristic is represented as an array that encodes the tile ranking, along with the quality level assigned to each of the tiles. Traditionally, a VR client would take said array and download each of the tiles, at the specified quality level, into the playout buffer. The prefetching mechanism we propose contemplates an extra step: forwarding

the rate adaptation result to the cache-enabled edge server as soon as it is generated, before the client starts buffering video content for a given segment.

Returning to the example introduced earlier, the output of the CTF rate adaptation heuristic in that case comprises six high-quality plus ten low-quality tiles, following the order indicated below in Figure 5.3. As the diagram illustrates, the client relays this output array to the *Prefetch server*. With this information, the specified tiles are requested concurrently from the *Content server*, taking advantage of a high-capacity link between them. Then, the corresponding video files are loaded into the cache memory, which serves the forthcoming requests from the client with low latency. This in turn should lead to an increase in the bandwidth perceived by the client, and as consequence, also in the quality of the content requested for subsequent video segments.

Figure 5.3 VR content prefetching: The output of the rate adaptation algorithm is fed to the prefetch server before the client's buffer starts filling up.



Clearly, conducting such a prefetching procedure for each individual user would entail a misuse of the cache memory resources and a substantial increase in the backhaul traffic and the content server's load. To address this issue, we propose a stream processing method for estimating the most salient tiles according to the viewers fixation patterns, on a per-video segment basis. Said set of per-segment salient tiles is then stored into the data structure we refer to as DCoB.

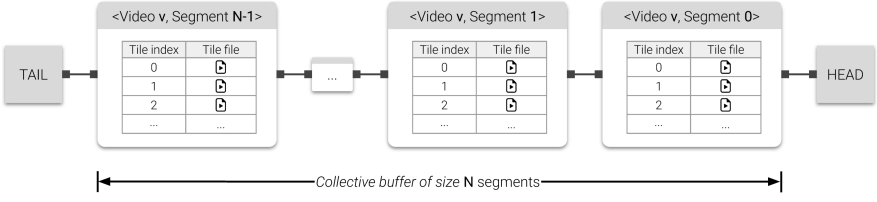
5.3.2 Dynamic Collective Buffer (DCoB)

The DCoB can be understood as a common playout buffer shared by active viewers consuming the same VR content at a certain point in time. Think about the scenario in which a content provider premieres a new episode of a popular show. Many viewers are likely to start a streaming session soon after the episode has been released. In such scenario, clients can benefit greatly

from a nearby cache serving content that has been previously requested by other users. Of course, to make the most of the limited memory resources, only a subset of the tiles per segment should be stored into this cache, i.e. those that are most likely to be consumed in ongoing streaming sessions. Arranged in this way, the data in the cache configures a *per-segment collective viewport* or *collective buffer* keeping content from the last N video segments consumed thus far.

This collective buffer has been modeled as a FIFO queue of limited size, backed by a hash table to allow for instantaneous retrieval (see Figure 5.4 below). Once the configured capacity is exceeded, the tiles corresponding to the least recently requested segment are evicted, freeing up space in memory for new segments.

Figure 5.4 Collective buffer as a FIFO queue. Each item in the queue corresponds to one segment of a given video and contains the most relevant tiles prefetched from the content server.



The set of video tiles contained within each of the segments of the collective buffer should be dynamically adjusted in response to viewers' fixation patterns. In Section 5.3.1, a tile ranking was obtained as output of the CTF rate adaptation heuristic. This ordered list of tiles encodes the estimated fixation map of an individual user when watching a particular video segment. In this sense, we have devised an incremental procedure that enables merging the ordered preferences of all the users with an active streaming session, into a single list of video tiles per segment, composing a collective fixation map.

Let us represent a viewer's fixation map for video v and segment s as:

$$\phi_{v,s} = \{\langle t, \rho(t)\rangle : t \in \{1, ..., m \cdot n\}\} \tag{5.1}$$

where t represents each of the $m \cdot n$ tiles per segment in the tiling scheme $(m \times n)$, while ρ is a function that returns the position in the viewer's tile ranking of the tile passed as argument. Considering the running example from the previous section (see Figure 5.3), the corresponding fixation map can be expressed in the following terms:

$$\begin{aligned}\phi_{v,s} = \{&\langle 1,14\rangle, \langle 2,8\rangle, \langle 3,4\rangle, \langle 4,8\rangle, \langle 5,13\rangle, \langle 6,7\rangle, \langle 7,1\rangle, \langle 8,3\rangle, \langle 9,15\rangle,\\ &\langle 10,9\rangle, \langle 11,2\rangle, \langle 12,6\rangle, \langle 13,16\rangle, \langle 14,12\rangle, \langle 15,11\rangle, \langle 16,10\rangle\}\end{aligned} \tag{5.2}$$

Now, to combine the fixation maps of K viewers watching segment s of video v, we start by computing the average position, ρ, for each video tile over all K-fixation maps. The collective fixation map ($\bar{\phi}_{v,s}$) is defined as follows:

$$\bar{\phi}_{v,s} = \left\{ \left\langle t, \frac{1}{K} \sum_{i=1}^{K} \rho_{\langle i \rangle}(t) \right\rangle : t \in \{1, ..., m \cdot n\} \right\} \tag{5.3}$$

The order of the tiles in $\bar{\phi}_{v,s}$ is determined by their average position, i.e. the smaller this value is for a certain tile, the higher the precedence the tile has for the given video segment.

As stated earlier, only a subset of these tiles should make it to the corresponding segment of the collective buffer. We refer to this subset as *collective viewport*, defined as the *top-k* tiles of the collective fixation map. To determine the value of k we first estimate the correlation between the viewers' fixation maps. High correlation between these maps would imply that users are looking at the same sections of the display, i.e. a few specific tiles. We estimate said correlation by using the *Kendall's tau coefficient* ($\mathcal{K}_\tau$) [41], which measures the correspondence between two ordered sequences in the range $[-1, 1]$: the closer to 1 (resp. -1) the higher (resp. lower) the correspondence. Finally, the value of k is set to be proportional to the complement of this correlation coefficient, which we refer to as *Kendall's tau distance* ($\mathcal{K}_{\tau dist}$). Let us take $\bar{\phi}_{_{curr}v,s}$ as the current collective fixation map for segment s of video v, and $\phi_{_u v,s}$ as a new fixation map corresponding to user u, for the same video segment. The collective viewport size, k, is computed as follows:

$$\begin{aligned} k &= \left\lceil m \cdot n \cdot \mathcal{K}_{\tau dist}(\bar{\phi}_{_{curr}v,s}, \phi_{_u v,s}) \right\rceil ; \\ \mathcal{K}_{\tau dist}(\bar{\phi}_{_{curr}v,s}, \phi_{_u v,s}) &= 1 - \frac{\mathcal{K}_\tau(\bar{\phi}_{_{curr}v,s}, \phi_{_u v,s}) + 1}{2} \end{aligned} \tag{5.4}$$

From the equations in 5.4, note that in case of perfect correlation ($\mathcal{K}_\tau = 1$), the distance between the fixation maps is zero ($\mathcal{K}_{\tau dist} = 0$), and therefore the viewport size, k, is equal to zero as well. In these circumstances, since both the *collective* and *new* fixation maps contain the same collection of tiles, the collective viewport stored into the DCoB for the given segment and video should remain unmodified.

The collective fixation map is incrementally refined as new viewers show up. For this the prefetch server keeps track of the number of viewers ($nViews$)

that have watched a given video segment, along with the per-segment cumulative Kendall's tau distance ($aggK_{\tau dist}$) computed across all the fixation maps received thus far. This data is kept in a key-value store with the tuple $\langle v, s \rangle$ being designated as key:

$$\mathcal{F} : (\langle v, s \rangle) \mapsto \left[\bar{\phi}_{v,s}, nViews, aggK_{\tau dist}\right] \tag{5.5}$$

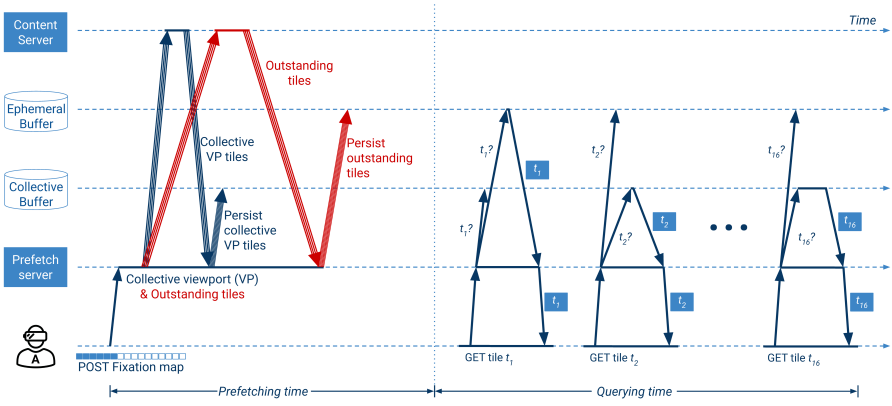
Algorithm 5.1 DYNAMIC COLLECTIVE BUFFER

1: Let $\mathcal{V}$ be a catalog of 360° videos
2: Let $\mathbf{\Phi}$ be a stream of fixation maps forwarded from the VR clients
3: $\mathbf{\Phi} = [\phi_{v,s} : v \in \mathcal{V} \wedge s \in \{1, 2, 3, \dots\}]$ ▷ Unbounded set of fixation maps
4: Let $m \cdot n$ be the number of tiles according to the tiling scheme ($m \times n$)
5: Let N the capacity of the collective buffer (number of video segments)
6: Let $\boldsymbol{DCoB}(N)$ be the collective buffer (a FIFO queue backed by a hash table)
7: $\mathcal{F} \leftarrow$ empty dictionary
8: **for each** fixation map $\phi_{v,s}$ in $\mathbf{\Phi}$ **do**
9: **if** $\langle v, s \rangle \notin \mathcal{F}.keys()$ **then** ▷ There is no fixation map for tuple $\langle v, s \rangle$ yet
10: $\bar{\phi}_{v,s} \leftarrow \phi_{v,s}$ ▷ Initialize *collective fixation map*
11: $nViews \leftarrow 1$
12: $aggK_{\tau dist} \leftarrow 0$
13: $k \leftarrow m \cdot n$ ▷ Initialize size k of the collective viewport to $m \cdot n$
14: **else**
15: $\bar{\phi}_{v,s}, nViews, aggK_{\tau dist} \leftarrow \mathcal{F}.get(\langle v, s \rangle)$
16: $\bar{\phi}_{v,s} \leftarrow mergeFixationMaps(\bar{\phi}_{v,s}, \phi_{v,s})$ ▷ Merge $\phi_{v,s}$ into *collective fixation map*
17: $nViews \leftarrow nViews + 1$
18: $aggK_{\tau dist} \leftarrow aggK_{\tau dist} + \mathcal{K}_{\tau dist}(\bar{\phi}_{v,s}, \phi_{v,s})$
19: $k \leftarrow \left\lceil \frac{aggK_{\tau dist}}{nViews}(m \cdot n) \right\rceil$ ▷ Set k to be proportional to average $\mathcal{K}_{\tau dist}$
20: **end if**
21: **if** $\boldsymbol{DCoB}.size() \geq N$ **then** ▷ If the buffer capacity has been exceeded then remove the segment at the head of the queue
22: $\langle v_H, s_H \rangle \leftarrow \boldsymbol{DCoB}.pop()$
23: $\mathcal{F}.remove(\langle v_H, s_H \rangle)$
24: **end if**
25: $\mathcal{F}.set(\langle v, s \rangle, \left[\bar{\phi}_{v,s}, nViews, aggK_{\tau dist}\right])$ ▷ Update key-value store with new values
26: **if** $k \neq 0$ **then**
27: $collectiveVP_{v,s} \leftarrow topK(\bar{\phi}_{v,s}, k)$ ▷ Collective viewport set to the top k tiles of $\bar{\phi}_{v,s}$
28: $VPVideoTiles_{v,s} \leftarrow HTTPGetFromContentServer(collectiveVP_{v,s})$
29: $\boldsymbol{DCoB}.set(\langle v, s \rangle, VPVideoTiles_{v,s})$ ▷ Update data at the *collective buffer*
30: **end if**
31: **end for**

The formal procedure for processing the stream of fixation maps coming from connected VR clients is specified in Algorithm 5.1. The process starts by first initializing $\mathcal{F}$ as an empty key-value store (line 7). Then the fixation maps $\phi_{v,s}$ are taken in, one after the other (line 8). Each fixation map updates its corresponding entry on the collective buffer. The *mergeFixationMaps* function in line 16 represents the incremental application of the operation referred earlier in equation 5.3. The output of this function is the collective fixation map modified by the fixation map being currently processed. The size of the collective viewport, k, is determined as the closest integer to the product of the average Kendall's tau distance ($\frac{aggK_{\tau dist}}{nViews}$)

times the total number of tiles ($m \cdot n$). This way the input from previous viewers is weighted and taken into account (line 19). Finally, the tiles belonging to the collective viewport are obtained (i.e. the first k tiles from the collective fixation map), the corresponding video files are retrieved from the *content server*, and the up-to-date data is stored into the collective buffer (***DCoB***) and the key-value store ($\mathcal{F}$) (lines 25-29), after ensuring that the maximum configured capacity (N) is not exceeded (lines 21-24).

Figure 5.5 Timeline of a typical interaction between the entities composing the VR content prefetching approach. At *prefetching time* the collective viewport and outstanding tiles are downloaded into the prefetch server memory. These tiles are served to the client with low latency at *querying time*.



Along with the collective buffer, we also defined a short-lived buffer into which the prefetch server stores the set of *outstanding tiles*, namely those tiles in the viewer's fixation map that remain outside the collective viewport. This in order to avoid the client having to wait for the content server to deliver these tiles during querying time, preventing playout freezes from happening. The entries in this *ephemeral buffer* are volatile and expire over a period of time equivalent to one video segment to minimize their memory footprint. Having both the collective and ephemeral buffers in place ensures that the client can always find relevant content loaded into the prefetch server memory. This way we manage to bypass the *cold-start* problem typical of traditional caching solutions. Figure 5.5 illustrates a typical sequence of interactions that take place between client, servers and data stores for a single viewer.

5.3.3 Analysis of computational cost

The procedure in charge of conducting content prefetching has been conceived as a *stateful streaming algorithm* (see Algorithm 5.1). The input of said procedure consists of regular array structures representing the viewer's fixation maps (consider the example in Equation 5.2). The length of these arrays is fixed and determined by the number of tiles of the tiling scheme in use, i.e., $m \cdot n$. The proposed algorithm processes each array on an individual basis, and the output of such a processing alters the state of a collective fixation map and the collective and ephemeral buffers, for a given video v and segment s. These data structures represent the state being managed by the algorithm. Let us consider the cost incurred in this procedure both in terms of space and time.

5.3.3.1 Space cost

As described earlier in Section 5.3.2, the data structures that maintain the state in the proposed algorithm are all arranged into hash tables persisted in memory to allow for fast read and write operations. The hash tables of both the key-value store holding the per-segment collective fixation maps ($\mathcal{F}$), and the collective buffer ($\boldsymbol{DCoB}$) have a fixed capacity in terms of the number of segments they can contain. Said capacity is set upfront via a configuration parameter N. In this sense, the space cost due to these two data structures is proportional to $O(N)$.
The hash table backing the ephemeral buffer stores individual tiles which are not part of the collective viewport for a given video and segment. In these circumstances, the space cost is proportional to the number of tiles the viewer is likely to watch in the upcoming segment that fall outside the collective viewport. Said number is never greater than $m \cdot n$ (worst-case scenario). Additionally, the video content persisted in this ephemeral buffer is short-lived by design, which further reduces its memory footprint.

5.3.3.2 Time cost

At the core of the procedure for maintaining the collective buffer lie two operations:

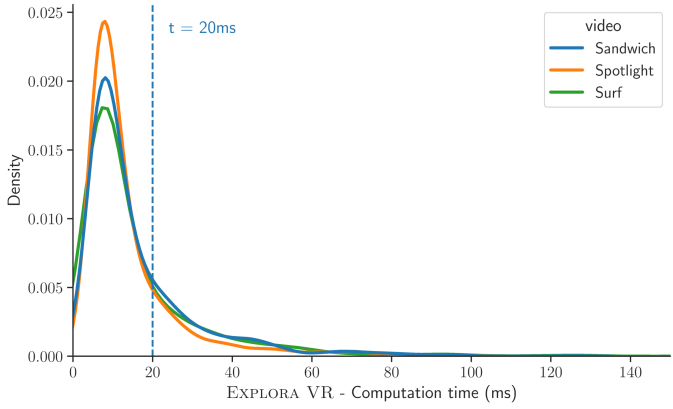
i. the function that updates the collective fixation map ($\bar{\phi}_{v,s}$) for a certain video v and segment s, taking in a new unseen fixation map ($\phi_{v,s}$) (see line 16 in Algorithm 5.1)

ii. the function that incrementally computes the Kendall's tau distance ($\mathcal{K}_{\tau dist}$) between the current $\bar{\phi}_{v,s}$ and the incoming $\phi_{v,s}$ (see line 18 in Algorithm 5.1)

The first operation consists of computing the element-wise average of two indexed arrays of size $m \cdot n$, and subsequently sorting the resulting array on the obtained values. By using an algorithm such as *mergesort*, the time it takes for this operation to run is proportional to $O(mn \log mn)$.

The Kendall's tau distance in the second operation is computed using the method by Knight [42], implemented in the `SciPy` Python library. This method is known to have linearithmic time complexity, which in this particular case translates to $O(mn \log mn)$, just as with the above-mentioned operation.

Figure 5.6 Experimental determination of the time required to compute the collective viewport. An in-depth description of the setup is provided in Section 5.5.1.



Since m and n values are fixed and typically small (consider for instance a 4×4 tiling scheme), the proposed algorithm is expected to feature a low and fairly consistent execution time. Figure 5.6 shows an example of the computation times measured on an experimental setting with 48 viewers watching the first 30 segments of three different 360° videos, using a 4×4 tiling scheme. In said setting (described in detail later in Section 5.5.1), the devised operations for computing the collective viewport run under 20 milliseconds 80% of the time. This is only $\frac{1}{50}$ to $\frac{1}{200}$ of the video segment length used in tile-based omnidirectional video streaming applications, which typically ranges between one to four seconds [16].

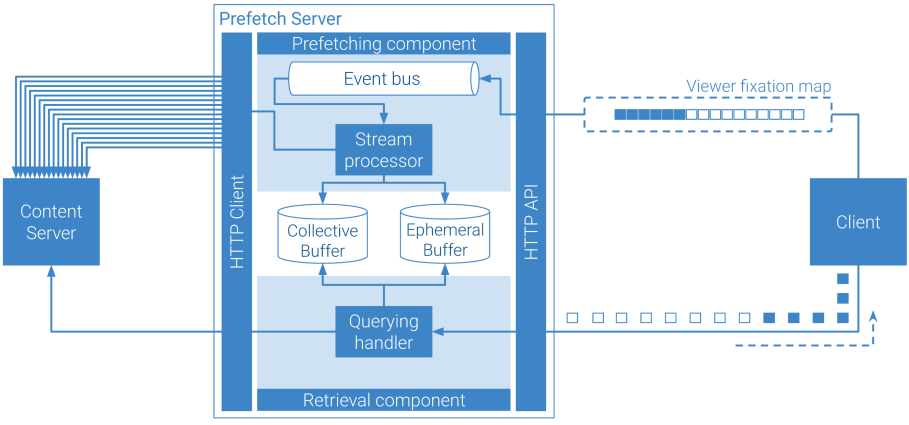
Note that the computational cost of the proposed mechanism largely depends on configuration parameters such as the collective buffer capacity (N) and the tiling scheme ($m \times n$). This suggests that, as the number

of users increases, memory use will not surge out of control and processing time will remain consistent, which accounts for the scalability of our content approach.

5.4 Architecture and Proof-of-Concept Implementation

The system that implements the content prefetching mechanisms we introduced in the previous section adopts an architecture featuring highly configurable containerized components. This system supports the emulation of multiple VR video streaming scenarios—with and without prefetching enabled—under different network and load conditions. A diagram of the components and submodules that make up the system is presented in Figure 5.7. Next, we address the description of the components of this architecture.

Figure 5.7 VR content prefetching architecture: inspired by the EXPLORA framework by Ordonez et al. [43]



5.4.1 Prefetch Server

This is the core component of the system. In devising the functional submodules of this server, we have drawn inspiration from the data processing pipeline presented by Ordonez et al. in [43], which decouples stream data ingestion/preprocessing from data storage and content retrieval. The prefetch server features three main modules: (*1*) the *prefetching component*, (*2*) the *content buffers*, and (*3*) the *retrieval component*.
The *prefetching component* provides an *event bus* which collects the viewers' fixation maps fed by the VR client. A *stream processor* in this component

consumes said fixation maps and runs the procedure specified earlier in Algorithm 5.1 to incrementally build the collective viewports. The stream processor is also in charge of fetching video content from the *Content server*, and does this by issuing multiple concurrent HTTP requests. We relied on the *Publish/Subscribe* pattern readily available in the *Redis* in-memory data store [44] to implement the event bus. As for the stream processor, we implemented it as a Python application running continuously in background, along with the *HTTP API* in charge of handling the interaction with the client.

The video content fetched from the content server by the stream processor is loaded into the *data buffers*. The *collective buffer* hosts the arrangement of video tiles lying inside the incrementally computed viewports, while the *ephemeral buffer* stores the outstanding tiles as defined at the end of Section 5.3.2. Both buffers are backed by key-value databases implemented in *Redis*.

The *retrieval component* implements the *querying handler* submodule in charge of processing clients' requests for video content. Upon receiving a query, this handler looks up the corresponding video tile file into both the collective and ephemeral buffers. In case the video file is not available yet in none of the prefetch buffers (e.g., due for instance to quality mismatch or network delay), the handler would relay the request to the content server.

The implementation of the prefetch server is available online at https://github.com/LeandroOrdonez/explora-vr-cache.

5.4.2 Content Server

This component plays the role of one of the nodes from a content delivery network (CDN). The *content server* consists of a containerized Web server publishing the tiled video content through a HTTP API. Video files are served from the local file system of this component in response to regular `HTTP/1.1 GET` requests matching the following the URL pattern:

```
http://<:host>:<:port>/<:video_id>/<:t_hor>x<:t_vert>/
    <:quality_id>/seg_dash_track<:tile_id>_<:segment_id>.m4s
```

where `t_hor` and `t_vert` stand for the number of tiles in the horizontal and vertical axes respectively, according to the applied tilling scheme.

This content server component was implemented as a Python Web application using the *Flask* framework and *NGINX+uWSGI* as application server. The code of this implementation is available online as well at https://github.com/LeandroOrdonez/explora-vr-server.

5.4.3 Client

This component is a containerized adaptation of the headless Virtual Reality client developed by van der Hooft et al. [32]. The headless VR client is an adaptive streaming application written in Python which is able to recreate video streaming sessions from prerecorded head movement traces. By deploying this component as an independent containerized application, we were able to spawn multiple concurrent video streaming sessions, allowing us to assess the response of the proposed VR video content prefetching mechanism under different network and load conditions. The code of the original implementation of the headless VR client is available at https://github.com/jvdrhoof/VRClient, while our adaptation can be found at https://github.com/LeandroOrdonez/explora-vr-dash-client.

5.5 Experimental Evaluation

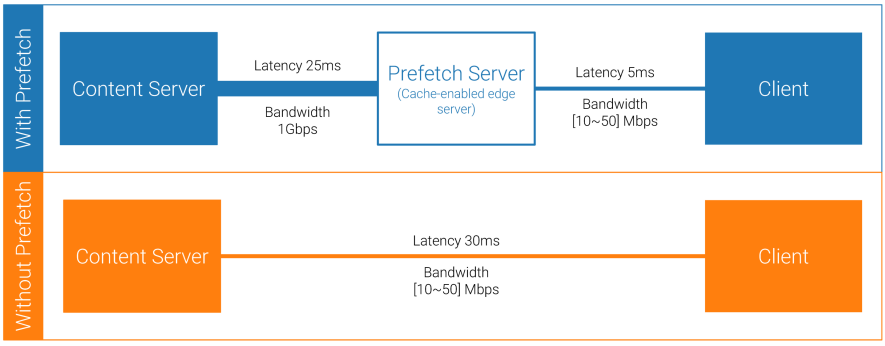
To determine the strengths, costs and limitations of the content prefetching mechanism, we have conducted a benchmark evaluation on various VR video streaming setups, with and without the prefetch mechanism in place. The prefetching approach presented in this article was also compared to a caching strategy with a traditional *least-recently used* (LRU) replacement policy which is a common baseline used for evaluating the performance of existing edge-assisted solutions. A description of the environment configuration and the covered test scenarios is presented next, along with the results obtained from this evaluation.

5.5.1 Experimental Setup

The experimental testbed we used in this evaluation is depicted in Figure 5.8. Each of the components in this diagram were deployed as an isolated *Docker* container, running on a single host machine with 20GB RAM, Intel E5645s @ 2.4GHz processor, and 54GB Hard Disk, using the infrastructure provided by the *imec/IDLab Virtual Wall* environment [45]. As is typically the case, we assume the link between the content server and the cache-enabled edge server to have higher capacity/higher latency than the one between the prefetch server and the VR clients. To emulate these conditions, we have run *traffic control* (`tc`) [46] on each of the containers. This way, we have provisioned a connection between content and prefetch server with 1 Gbps bandwidth capacity and 25 milliseconds latency. On the client's end, we set the latency to 5 milliseconds for the setup with prefetching enabled, and 30 milliseconds in the setup without prefetching—i.e. we kept the same round trip time (RTT) between client and content server in both setups.

We gradually increased the bandwidth in the clients link from 10 Mbps to

Figure 5.8 Experimental testbed for evaluating the VR content prefetching mechanism



50 Mbps, and estimated the impact the devised prefetching mechanism has on the quality of experience (QoE) perceived by the user, measured in terms of delivered video quality, startup delay, and occurrence of playout freezes, as reported by the VR client on a per-segment basis. Finally, these results are contrasted to those obtained from a setup implementing a traditional LRU replacement policy in the cache-enabled edge server.

As for the video content we used the dataset created by Wu et al. [21], which provides head movement traces recorded from 360° video streaming sessions. This dataset comprises the traces collected from 48 unique users while watching nine different VR videos. The tests run in this evaluation consider three representative videos out of the original nine: *Sandwich* features a fragment of a talk show in which most of the motion concentrated in the center of the display; *Spotlight* presents a more dynamic sequence typical for an action movie; *Surf* displays a compilation of video clips recorded with a GoPro camera in an open environment. A tiling scheme of 4×4 was applied to each of these videos at 4K resolution and 30 FPS, using the same encoder and parameters discussed in [32] and listed in Table 5.1. We used two quality levels to encode each of the three videos, corresponding to constant rate factors (CRF) of 15 (*High quality*) and 35 (*Low quality*). Table 5.2 summarizes the resulting bitrates for both quality representations.

With this setup in place, we proceeded to emulate a scenario with multiple users connecting to a video streaming event. In this scenario, each of the 48 viewers in the dataset by Wu et al. [21] would start a streaming session to watch the first 30 segments—this is 32 seconds for a segment duration of 1.067 seconds—of each of the three considered videos. In order to approximate the dynamics of such *near-live on-demand* streaming scenario serving

Table 5.1: Overview of encoding parameters

Parameter	**Value**
Encoder	HEVC Test Model (HM)
Tiling scheme	4×4 at 4K resolution and 30 FPS
GOP	32
Segment duration	≈1.067s
CRF	[15, 35]

Table 5.2: Quality levels and corresponding bitrates for the three videos.

Video	Bitrate [Mbps]	
	High Quality	Low Quality
Sandwich	21.9±6.6	1.2±0.3
Spotlight	20.8±13.9	1.4±1.3
Surf	26.4±12.7	2.4±1.4

multiple users, we set up the experiment so that viewers arrive to their watching session in quick succession with a 5 second separation between each other. This means there were no more than six users watching the same video at a given time.

We have run this simulation for three different configurations: (*i*) `NO_PREFETCH`: no prefetching/cache enabled, (*ii*) `PREFETCH`: prefetching enabled with a collective buffer of 30 segments in size, and (*iii*) `LRU`: caching with LRU replacement strategy and cache size limited to 70MB, which is slightly above the maximum value of memory used by the prefetching mechanism throughout the experiment, as shown below in Table 5.3. For each configuration, we measured the performance of the system in terms of segment download time, user's QoE (i.e., video quality, startup time, and occurrence of playback freezes), network traffic between content and edge server, and accuracy of the prefetch buffer/cache.

Table 5.3: Memory consumed by the prefetching and caching strategies

Bandwidth client's link (Mbps)	**Edge Server memory use (MB)**	
	`LRU`	`PREFETCH`
10	70	57.91
15	70	59.25
20	70	61.11
25	70	62.69
30	70	67.04
35	70	64.01
40	70	67.30
45	70	67.46
50	70	66.29

Finally, the versions of the software tools used in this evaluation are listed in Table 5.4.

Table 5.4: Versions of the software used in the experimental setup.

Software	**Version**
Docker	20.10.6, build 370c289
Docker Compose	1.17.1
Operating System	Ubuntu 18.04.4 LTS
Redis server	5.0.3
NGINX (*content and prefetch* servers)	1.14.2
uWSGI (*content and prefetch* servers)	2.0.17.1
Flask (*content and prefetch* servers)	1.0.2

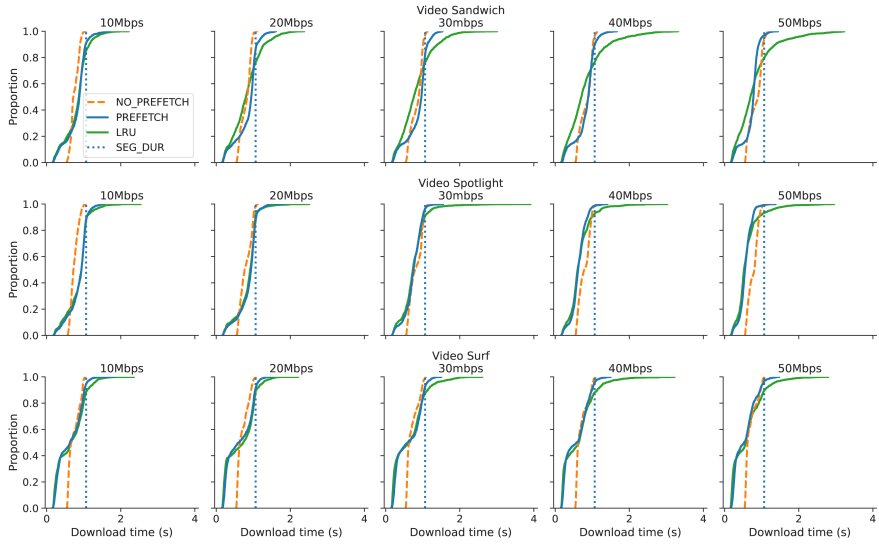
5.5.2 Results

The playout buffer size in VR video streaming is limited to a few segments to allow for fast adaptation to viewport changes. In this sense, these kind of streaming applications are particularly susceptible to buffer starvation and playout freezes. In a setup with a cache-enabled edge server placed between clients and the content server, the rate adaptation heuristic might be tricked into believing that content is closer than it actually is, which leads it to request video tiles in high-quality representations. In case of *cache misses* (i.e. the requested content is not found in the cache's memory) the request has to be relayed back to the server, which entails additional processing and network latency. In said cases, the segment download time might take longer than the segment playback duration. When such conditions persist for several segments during a watching session, buffer draining-out and playout freezes are bound to happen.

Figure 5.9 shows the empirical cumulative distribution function (ECDF) of the per-segment download time for the three configurations under evaluation (`NO_PREFETCH`, `PREFETCH`, and `LRU`), measured for multiple values of bandwidth on the client's end. Note that both the setup with the proposed prefetching mechanism, as well as the one with the LRU cache replacement policy manage to keep download times under the segment duration limit (`SEG_DUR` line in Figure 5.9) for most of the segments across all bitrates and videos. However, for the `LRU` setup, there is in general a larger proportion of segments taking longer to download than the segment duration: on average 14% of the segments in the `LRU` configuration, compared to only 7.6% of the segments in the `PREFETCH` setup. As the capacity on the client's link increases, those segments can take as much as 3.9 seconds to download, which is far higher than the comparable download times from the `PREFETCH` setup which do not surpass 1.8 seconds in any of the cases. This signals a

higher likelihood of cache misses for the LRU configuration, and a more frequent occurrence of playout freezes in this setup, specially for large values of bandwidth on the client's connection.

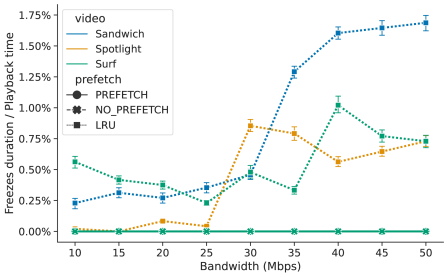
Figure 5.9 ECDF of the per-segment download time for the three tested configurations. The larger the number of segments taking longer than SEG_DUR to download, the more likely playout freezes are to occur.



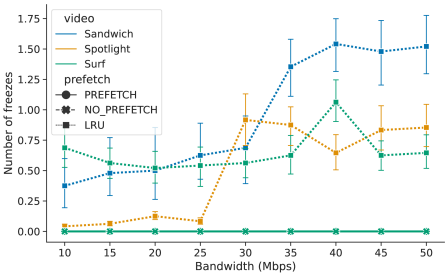
The foregoing is confirmed by measuring the number and duration of the playout freezes by streaming session. Figure 5.10 reports on these measurements as a function of the client's bandwidth, for each of the considered videos. Note that the setup with the proposed prefetch mechanism offers a *freeze-free* playback experience to the user, in contrast to the LRU counterpart. According to Figure 5.10a, the average number of freezes per streaming session on the LRU configuration is always greater than zero, and the number increases for the three videos as the bandwidth grows larger. We can observe a similar behavior for the total freeze duration. Figure 5.10b presents this measurement as a proportion of the length of a streaming session, i.e. 32 seconds. These results are clearly inconvenient and counterintuitive from the client's perspective, and can be attributed to the occurrence of cache misses. Table 5.5 below shows the cache hit ratio measured across the streaming sessions of all 48 users in the dataset, for both LRU and PREFETCH configurations. For the setup with content prefetching enabled, the hit ratio stays above 98% through the entire range of bandwidths, while for the configuration with LRU cache replacement it consistently decreases from 94.6% to 87% as the bandwidth increases. As the bandwidth in the client's link

grows larger, the quality of the requested content tends to increase, as does the size of the video tiles stored in the cache. In these circumstances, the LRU cache is only able to accommodate a few items, which in consequence increases the frequency of eviction cycles and cache misses.

Figure 5.10 Occurrence and duration of playout freezes: The `PREFETCH` and `NO_PREFETCH` configurations manage to deliver a freeze-free watching experience to the viewer. For the LRU setup, both frequency and duration of playout freezes increase as the bandwidth on the client's link grows larger.



(a) Average number of freezes per watching session.



(b) Average freeze duration to playback time ratio per streaming session.

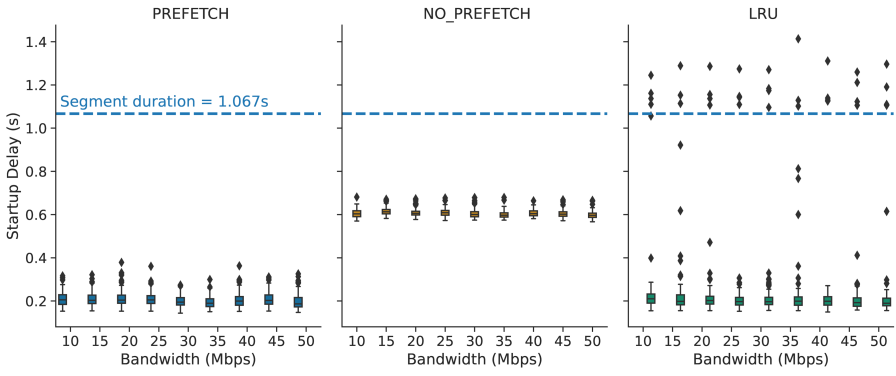
Table 5.5: Hit ratio for different values of bandwidth in the client's link.

Bandwidth client's link (Mbps)	**Hit ratio**	
	`LRU`	`PREFETCH`
10	94.62%	98.44%
15	93.84%	98.43%
20	92.56%	98.36%
25	91.65%	98.51%
30	89.40%	98.52%
35	88.17%	98.60%
40	87.37%	98.69%
45	87.27%	98.97%
50	87.02%	99.13%

Cache misses also occur as a consequence of the *cold-start* problem that affects passive caching strategies such as LRU. Requests issued against a cold cache are likely to be cache misses and therefore result in retrieval from the origin server. This leads to longer startup delays which degrade the QoE mainly for early viewers. Figure 5.11 presents the startup delay observed across all the streaming sessions as a function of the client's link capacity. Delay values remain relatively invariable as the bandwidth on the client's connection increases for all the considered configurations. Note that for both `PREFETCH` and `LRU` setups (left and right side in Fig. 5.11, respectively), the majority of the values are clustered around 200 millisec-

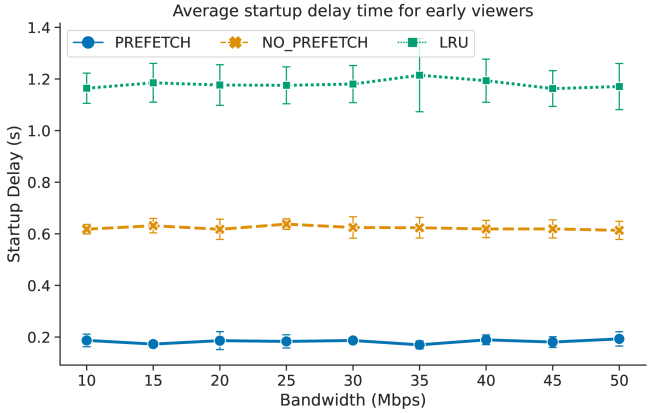
onds approximately. This represents a reduction of nearly 3× the startup delay viewers experience in the setup without prefetching/caching enabled (middle chart in Fig. 5.11). However, a large number of outliers is observed for the `LRU` configuration lying beyond the segment duration limit. This indicates that many viewers would experience more than one second latency from the moment they initiate the streaming session to the moment the video playback starts. These outliers represent the startup delay perceived by the first users as a consequence of their request hitting a cold cache and being relayed back to the content server. By forwarding said requests to the origin server, early users of the `LRU` setup incur an extra network hop which leads to startup delay times higher than those observed for the `NO_PREFETCH` setup. In Figure 5.12 only the startup latency measured for the group of early viewers is plotted. On average, early users in the `LRU` configuration would observe around 2× and 6× longer delay times compared to viewers in the `NO_PREFETCH` and `PREFETCH` setups, respectively. These results show that the proposed content prefetching mechanism is able to bypass the *cold-start* problem and offer not only shorter startup delay times but also a more consistent experience across all viewers compared to the alternative configurations.

Figure 5.11 Startup delay distribution as a function of the client's link capacity. Observations for `PREFETCH` and `LRU` configuration are largely concentrated around comparable values. However, outliers for the `LRU` setup lie farther apart from the bulk of the data, beyond the segment length in many cases. In comparison, the `PREFETCH` configuration offers a more consistent experience for all viewers.



So far, the proposed prefetching mechanism has proven able to deliver a user experience that outperforms that of the alternative setups in terms of segment download time, frequency/duration of playout freezes and startup latency. Let us now look into the perceived video quality. In the HAS

Figure 5.12 Startup delay times observed by early users: On average viewers in the `PREFETCH` configuration would experience 6.5× and 3.4× lower latency than those in the `LRU` and `NO_PREFETCH` setups, respectively.



client, quality level for each tile in a video segment is determined based on the *perceived bandwidth*, estimated as the quotient between the amount of bits downloaded per segment and the per-segment download time:

$$perceived\_bandwidth(s_i) = \frac{size(s_i)}{download\_time(s_i)} \tag{5.6}$$

In the expression above, the size of the segment (s_i) is proportional to the quality level of the tiles it comprises. This way, the perceived bandwidth provides a reliable indication of the video quality as observed by the user. Figure 5.13 shows the average perceived bandwidth over all watching sessions per video, as a function of the actual bandwidth on the client's link. The configuration with content prefetching enabled outperforms the `LRU` setup, most remarkably along the largest values of bandwidth. With the proposed mechanism running on a cache-enabled edge server, clients perceive on average up to 2.5× more link capacity in comparison to the configuration without prefetching, and up to 1.4× compared to the `LRU` configuration. This results in a higher number of tiles being downloaded in high quality.

Figure 5.14 presents the distribution of the amount of high-quality tiles per segment across the three videos. The mass of the distributions corresponding to each of the setups shifts towards the right (higher number of HQ tiles)

Figure 5.13 Client perceived bandwidth as a function of the actual link capacity. User experience greatly benefits from prefetching VR video content into a nearby server.

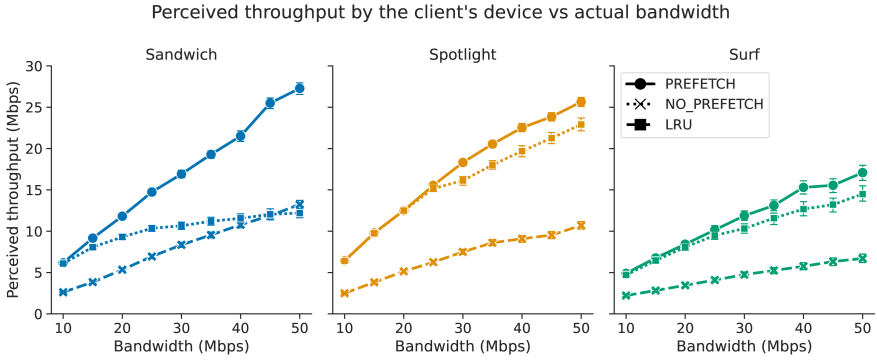
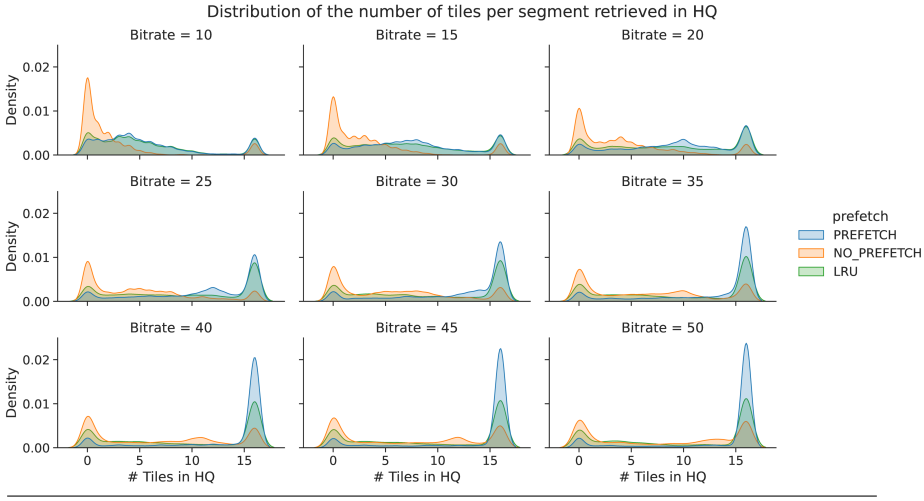


Figure 5.14 Distribution of the number of HQ tiles per segment: In comparison to the `LRU` and `NO_PREFETCH` configurations, the number of tiles retrieved in HQ from the prefetch setup increases more rapidly as the bandwidth grows larger. Bitrate values in the charts are in Mbps.



as the bandwidth increases. Note that for the configuration with prefetching enabled, the distribution tends to gravitate around 16 tiles/segment at a faster pace than the other two configurations. This proves that across all tested scenarios, the mechanism we propose consistently delivers higher quality of experience for the viewer, compared to the LRU cache alternative, and the plain vanilla client-server configuration.

Another appealing effect of prefetching and caching video content into an edge server is the reduction of network traffic to and from the content site. Table 5.6 presents the network traffic (in gigabytes) measured in the content server interface for the configuration without prefetching/caching enabled, along with the relative change of this metric for the LRU and PREFETCH setups, and how these measurements vary as the bandwidth on the client's link increases (see Figure 5.15 for the absolute values). Note that, thanks to the reuse enabled by the LRU and prefetching configurations, there is an important reduction in traffic to the content server: from 75% to 84% for LRU caching, and from 36% to 83% for the prefetching server. Also, it is worth noting that the setup with the proposed prefetching mechanism enables these network traffic savings while serving the highest video quality among the tested configurations. That is to say, serving a comparable video quality directly from the content server—without any prefetch/cache capabilitites—would require several times the network traffic reported in Table 5.6.

Table 5.6: Network traffic between content and prefetch servers for different values of bitrate in the client's link.

Bandwidth client's link (Mbps)	NO_PREFETCH **network traffic (GB)**	**% network traffic reduction**	
		LRU	PREFETCH
10	1.11	-75.68%	-36.04%
15	1.64	-78.66%	-44.51%
20	2.24	-80.36%	-48.21%
25	2.82	-82.27%	-53.90%
30	3.43	-80.76%	-64.14%
35	3.89	-81.75%	-64.78%
40	4.24	-82.31%	-67.45%
45	4.57	-83.15%	-80.96%
50	5.01	-84.23%	-83.03%

To understand why the LRU configuration results in a higher reduction of network traffic with respect to the implementation of the proposed prefetching mechanism, consider the fact that the latter setup is able to consistently deliver higher video quality levels than the former one throughout the entire range of bandwidth values. An increase in the capacity of the client's connection leads to a corresponding increase in the network throughput. This in turn prompts the client to request video tiles in higher quality representations, which consequently drives up the network traffic consumption. Figure 5.16 portrays the relation between bandwidth at the client side, network traffic in the content server's link, and video quality in terms of the number of high-quality tiles per segment delivered to the client. Note that while the setup with the LRU cache replacement strategy gets the upper hand with regard to network traffic reduction, the enhanced video quality added to the smooth playback offered by the proposed prefetching mecha-

nism, makes for a far superior QoE for the viewer. In this sense, the increase in network traffic to the content server for this configuration can be regarded as a reasonable price to pay.

Figure 5.15 Network traffic to the *content server* for the three considered configurations, as a function of the bandwidth in the client's link: Both the `LRU` and `PREFETCH` setups manage to induce a notable decline in network traffic. The prefetching mechanism enables this while delivering the highest quality of experience among the considered configurations.

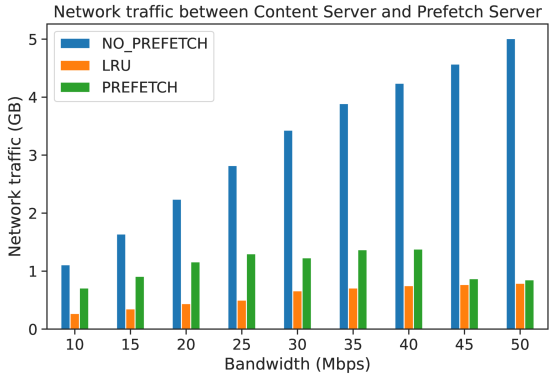
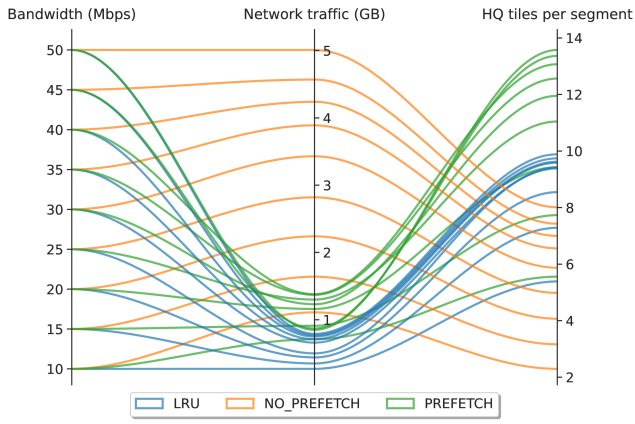


Figure 5.16 Relation between client's link bandwidth, network traffic in the backhaul link, and video quality. Both `LRU` and `PREFETCH` setups drive backhaul traffic down while increasing the number of tiles per segment served in high quality. The increase in network traffic use for the `PREFETCH` setup in relation to the `LRU` configuration obeys to a corresponding increase in the delivered video quality.



5.6 Conclusions

Immersive video applications are known for having an immense potential in sectors such as entertainment, education, healthcare, and digital services, among others. However, the existing network infrastructure still struggles to meet the stringent latency and bandwidth requirements of these kind of services, which remains a barrier to enable their broad adoption. In this chapter we presented EXPLORA-VR, an edge-assisted solution that allows for low-latency video streaming for tile-based immersive content.

EXPLORA-VR thrives on prefetching the tiles that users are likely to watch in the upcoming segments by advertising the outcome of the viewport prediction and rate adaptation algorithms, before the client starts consuming the content. Prefetched video tiles are downloaded to a cache-enabled edge server located in close proximity to the user, allowing for low-latency content retrieval. This in turn increases the link capacity perceived at the client's end, and in consequence also the quality level of the requested video tiles.

Additionally, the proposed solution supports content prefetching for an *on-demand*, *near-live* scenario, i.e. serving multiple active watching sessions streaming the same content within a narrow time window. To prevent the system from overflowing the content server with duplicate requests while doing this, EXPLORA-VR features a stream processing mechanism that incrementally builds a *collective playout buffer* to serve the requests from active users. This collective buffer is an in-memory data structure storing a fixed number of *collective viewports*, namely the group of tiles viewers tend to fixate the most on a per-segment basis. The per-segment collective viewports are continuously updated as new viewers arrive to dynamically accommodate to changes in the current preferences from the audience.

We evaluated the performance of EXPLORA-VR against a conventional *client-server* setup with no support for caching or prefetching, and an edge-assisted configuration implementing a regular LRU cache replacement strategy. Our solution proved to be effective in providing a smooth video playback, while also increasing the quality of the delivered content. Under equivalent network conditions, the devised prefetching mechanism leads to an average increase of 2.5× and 1.4× in the effective bandwidth perceived at the client's device compared to the conventional client-server and LRU setups, respectively. This in turn results in a proportional increase in the number of viewport tiles served in high quality. Moreover, in contrast to the alternative LRU configuration, our solution can consistently serve more than 98% of the content requests from the edge server. This means that only a minor proportion of the client requests get relayed to the origin content server, resulting in a freeze-free playback experience for the user. The

foregoing also signals the ability of the proposed approach to bypass the *cold-start* problem that typically affects passive caching strategies. The observed startup delay times show that EXPLORA-VR consistently provides low startup latency for all users, including early viewers. These results also hint at the potential of the proposed solution to aid in the recovery from eventual playback freezes. The proximity of the edge server coupled with the high prefetch hit ratio ensures that viewers can quickly resume the playback with a delay we expect to be comparable with the observed startup latency. Additional evaluations with real network traces are needed to confirm this assumption. The devised collective buffer also proved efficient in reducing the load on the content server network. Even though the LRU cache replacement policy outperforms the prefetching mechanism regarding this metric, the superior quality of experience that our approach can offer to the viewer reasonably outweighs this drawback.

In developing EXPLORA-VR, we assumed a number of conditions that will be relaxed in future work to make this solution more suitable for a production-level VR video streaming service. In this sense, further research is going to explore the effect of working with a lossy wireless network in the performance of the content prefetching mechanism. Likewise, the proposed solution will be extended to support multiple intermediate quality representations, instead of only *low-quality* and *high-quality* levels. We expect the results of this work will motivate further studies on edge-assisted prefetching techniques for omnidirectional video streaming.

References

[1] Jonathan Yépez, Luis Guevara, and Graciela Guerrero. AulaVR: Virtual Reality, a Telepresence Technique Applied to Distance Education. In *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–5. IEEE, 2020.

[2] Andrei OJ Kwok and Sharon GM Koh. COVID-19 and extended reality (XR). *Current Issues in Tourism*, pages 1–6, 2020.

[3] Ravi Pratap Singh, Mohd Javaid, Ravinder Kataria, Mohit Tyagi, Abid Haleem, and Rajiv Suman. Significant Applications of virtual Reality for COVID-19 Pandemic. *Diabetes & Metabolic Syndrome: Clinical Research & Reviews*, 14(4):661–664, 2020.

[4] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2019.02.050. URL https://www.sciencedirect.com/science/article/pii/S0167739X18319903.

[5] Shu Shi, Varun Gupta, Michael Hwang, and Rittwik Jana. Mobile VR on Edge Cloud: a Latency-Driven Design. In *Proceedings of the 10th ACM Multimedia Systems Conference*, pages 222–231, 2019.

[6] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. Latency and Cybersickness: Impact, Causes and Measures. A Review. *Frontiers in Virtual Reality*, 1:31, 2020.

[7] Maria Torres Vega, Christos Liaskos, Sergi Abadal, Evangelos Papapetrou, Akshay Jain, Belkacem Mouhouche, Gökhan Kalem, Salih Ergüt, Marian Mach, Tomas Sabol, et al. Immersive Interconnected Virtual and Augmented Reality: A 5G and IoT Perspective. *Journal of Network and Systems Management*, 28(4):796–826, 2020.

[8] Abid Yaqoob, Ting Bi, and Gabriel-Miro Muntean. A Survey on Adaptive 360° Video Streaming: Solutions, Challenges and Opportunities. *IEEE Communications Surveys & Tutorials*, 22(4):2801–2838, 2020.

[9] Dongbiao He, Cédric Westphal, and JJ Garcia-Luna-Aceves. Network Support for AR/VR and Immersive Video Application: A Survey. In *ICETE (1)*, pages 525–535, 2018.

[10] Kaixuan Long, Ying Cui, Chencheng Ye, and Zhi Liu. Optimal Wireless Streaming of Multi-Quality 360 VR V by Exploiting Natural, Rel-

ative Smoothness-enabled and Transcoding-enabled Multicast Opportunities. *IEEE Transactions on Multimedia*, 2020.

[11] Lingzhi Zhao, Ying Cui, Zhi Liu, Yunfei Zhang, and Sheng Yang. Adaptive Streaming of 360 Videos With Perfect, Imperfect, and Unknown FoV Viewing Probabilities in Wireless Networks. *IEEE Transactions on Image Processing*, 30:7744–7759, 2021.

[12] Jeroen van der Hooft, Maria Torres Vega, Stefano Petrangeli, Tim Wauters, and Filip De Turck. Optimizing Adaptive Tile-Based Virtual Reality Video Streaming. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 381–387. IEEE, 2019.

[13] Jeroen van der Hooft, Maria Torres Vega, Stefano Petrangeli, Tim Wauters, and Filip De Turck. Tile-Based Adaptive Streaming for Virtual Reality Video. *ACM Trans. Multimedia Comput. Commun. Appl.*, 15(4), December 2019. ISSN 1551-6857. doi: 10.1145/3362101. URL https://doi.org/10.1145/3362101.

[14] Mohammad Hosseini. View-Aware Tile-based Adaptations in 360 Virtual Reality Video Streaming. In *2017 IEEE Virtual Reality (VR)*, pages 423–424. IEEE, 2017.

[15] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 360probdash: Improving QoE of 360 Video Streaming Using Tile-Based HTTP Adaptive Streaming. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 315–323, 2017.

[16] Mario Graf, Christian Timmerer, and Christopher Mueller. Towards Bandwidth Efficient Adaptive Streaming of Omnidirectional Video Over HTTP: Design, Implementation, and Evaluation. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 261–271, 2017.

[17] Duc V Nguyen, Huyen TT Tran, Anh T Pham, and Truong Cong Thang. An Optimal Tile-Based Approach for Viewport-Adaptive 360-Degree Video Streaming. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):29–42, 2019.

[18] Wen-Chih Lo, Ching-Ling Fan, Jean Lee, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 360 Video Viewing Dataset in Head-Mounted Virtual Reality. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 211–216, 2017.

[19] Erwan J David, Jesús Gutiérrez, Antoine Coutrot, Matthieu Perreira Da Silva, and Patrick Le Callet. A Dataset of Head and Eye Movements for 360 Videos. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 432–437, 2018.

[20] Stephan Fremerey, Ashutosh Singla, Kay Meseberg, and Alexander Raake. AVTrack360: An Open Dataset and Software Recording People's Head Rotations Watching 360° Videos on an HMD. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 403–408, 2018.

[21] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. A Dataset for Exploring User Behaviors in VR Spherical Video Streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, page 193–198, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350020. doi: 10.1145/3083187.3083210. URL https://doi.org/10.1145/3083187.3083210.

[22] Silvia Rossi, Cagri Ozcinar, Aljosa Smolic, and Laura Toni. Do Users Behave Similarly in VR? Investigation of the User Influence on the System Design. *ACM Trans. Multimedia Comput. Commun. Appl.*, 16 (2), May 2020. ISSN 1551-6857. doi: 10.1145/3381846. URL https://doi.org/10.1145/3381846.

[23] Georgios Papaioannou and Iordanis Koutsopoulos. Tile-Based Caching Optimization for 360° Videos. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Mobihoc '19, page 171–180, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367646. doi: 10.1145/3323679.3326515. URL https://doi.org/10.1145/3323679.3326515.

[24] Anahita Mahzari, Afshin Taghavi Nasrabadi, Aliehsan Samiei, and Ravi Prakash. Fov-aware edge caching for adaptive 360 video streaming. In *Proceedings of the 26th ACM international Conference on Multimedia*, pages 173–181, 2018.

[25] Pantelis Maniotis and Nikolaos Thomos. Viewport-Aware Deep Reinforcement Learning Approach for 360 Video Caching. *IEEE Transactions on Multimedia*, 2021.

[26] Niklas Carlsson and Derek Eager. Had You Looked Where I'm Looking? Cross-User Similarities in Viewing Behavior for 360-Degree Video and Caching Implications. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page

130–137, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450369916. doi: 10.1145/3358960.3379129. URL https://doi.org/10.1145/3358960.3379129.

[27] Jianmei Dai, Zhilong Zhang, Shiwen Mao, and Danpu Liu. A View Synthesis-Based 360° VR Caching System over MEC-enabled C-RAN. *IEEE Transactions on Circuits and Systems for Video Technology*, 30 (10):3843–3855, 2019.

[28] Kedong Liu, Yanwei Liu, Jinxia Liu, Antonios Argyriou, and Ying Ding. Joint EPC and RAN Caching of Tiled VR Videos for Mobile Networks. In *International Conference on Multimedia Modeling*, pages 92–105. Springer, 2019.

[29] Shunyi Wang, Xiaobin Tan, Simin Li, Xiang Xu, Jian Yang, and Quan Zheng. A QoE-based 360° Video Adaptive Bitrate Delivery and Caching Scheme for C-RAN. In *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, pages 49–56. IEEE, 2020.

[30] Zheng Chang, Lei Lei, Zhenyu Zhou, Shiwen Mao, and Tapani Ristaniemi. Learn to Cache: Machine Learning for Network Edge Caching in the Big Data Era. *IEEE Wireless Communications*, 25(3):28–35, 2018.

[31] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. An HTTP/2-Based Adaptive Streaming Framework for 360° Virtual Reality Videos. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, page 306–314, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349062. doi: 10.1145/3123266.3123453. URL https://doi.org/10.1145/3123266.3123453.

[32] J. van der Hooft, M. Torres Vega, S. Petrangeli, T. Wauters, and F. De Turck. Quality Assessment for Adaptive Virtual Reality Video Streaming: A Probabilistic Approach on the User's Gaze. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 19–24, 2019. doi: 10.1109/ICIN.2019.8685904.

[33] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 Video Delivery over Cellular Networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, ATC '16, page 1–6, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342490. doi: 10.1145/2980055.2980056. URL https://doi.org/10.1145/2980055.2980056.

[34] Zhimin Xu, Xinggong Zhang, Kai Zhang, and Zongming Guo. Probabilistic Viewport Adaptive Streaming for 360-Degree Videos. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[35] Yuanxing Zhang, Pengyu Zhao, Kaigui Bian, Yunxin Liu, Lingyang Song, and Xiaoming Li. DRL360: 360-degree Video Streaming with Deep Reinforcement Learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1252–1260. IEEE, 2019.

[36] Johanna Vielhaben, Hüseyin Camalan, Wojciech Samek, and Markus Wenzel. Viewport Forecasting in 360° Virtual Reality Videos with Machine Learning. In *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 74–747. IEEE, 2019.

[37] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Fixation Prediction for 360 Video Streaming in Head-Mounted Virtual Reality. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 67–72, 2017.

[38] Yucheng Zhu, Guangtao Zhai, and Xiongkuo Min. The prediction of head and eye movement for 360 degree images. *Signal Processing: Image Communication*, 69:15–25, 2018.

[39] Vincent Sitzmann, Ana Serrano, Amy Pavel, Maneesh Agrawala, Diego Gutierrez, Belen Masia, and Gordon Wetzstein. Saliency in VR: How do people explore virtual environments? *IEEE transactions on visualization and computer graphics*, 24(4):1633–1642, 2018.

[40] Xinwei Chen, Ali Taleb Zadeh Kasgari, and Walid Saad. Deep Learning for Content-Based Personalized Viewport Prediction of 360-degree VR Videos. *IEEE Networking Letters*, 2(2):81–84, 2020.

[41] Roshani K Prematunga. Correlational Analysis. *Australian Critical Care*, 25(3):195–199, 2012.

[42] William R Knight. A Computer Method for Calculating Kendall's tau with Ungrouped Data. *Journal of the American Statistical Association*, 61(314):436–439, 1966.

[43] Leandro Ordonez-Ante, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert, and Filip De Turck. EXPLORA: Interactive Querying of Multidimensional Data in the Context of Smart Cities. *Sensors*, 20(9): 2737, 2020.

[44] Felipe Gutierrez. Messaging with Redis. In *Spring Boot Messaging*, pages 81–92. Springer, 2017.

[45] imec/IDLab. Virtual Wall: Perform large networking and cloud experiments., 2021. URL https://doc.ilabt.imec.be/ilabt/virtualwall/index.html.

[46] Martin A Brown. Traffic Control howto. *Guide to IP Layer Network*, 49:36, 2006.

6

Conclusions and Future Research Directions

"Problems worthy of attack prove their worth by hitting back."

—Piet Hein (1905–1996)

Data is profoundly imbued into nearly every aspect of modern society. As data becomes increasingly more pervasive, we strive to find even more innovative ways to harness it to augment our lives, make more of our resources, and back up the decisions we make. However, the sheer amount of data available nowadays, added to the accelerated rate at which it is being generated, exceeds the capacity of people and organizations to assimilate and make sense of it. To put it another way: most of the data that is produced today goes underused. Despite the rapid pace at which data technologies are being developed, accessing large, high-dimensional, and often distributed data remains an expensive and time-consuming task. This dissertation investigates several challenges concerning the provisioning of efficient querying mechanisms in big data environments. Four questions shaped the direction of the research presented in this thesis, as enumerated below:

RQ1: *How can the structure of a large dimensional dataset be dynamically adjusted to ensure instantaneous resolution of recurrent analytical queries?*

RQ2: *How can common exploratory analysis tasks be supported on live streams of multidimensional data under interactive (low-latency) time constraints?*

RQ3: *How can a balanced trade-off between server and client-side computation be reached to enable interactive exploration applications to serve requests from a large number of users?*

RQ4: *How can data retrieval patterns be identified in a streaming setting to dynamically enhance the user experience of multiple concurrent clients served by a latency-sensitive application?*

This dissertation introduces a number of solutions that contribute to addressing the formulated questions in various application domains including business analytics, smart cities, and content delivery for omnidirectional video streaming. The sections below summarize these contributions.

6.1 A workload-aware method for automatic view selection on large dimensional datasets

In line with the first research question, this method was conceived as a realization of a dynamic framework for schema transformation in dimensionally model datasets. Said framework, also formulated within the scope of the research documented in this dissertation, intends to incrementally alter or augment the existing dataset schema so that recurrent and time-consuming analytical queries can be resolved in far less time. In this way, for instance, the framework might decide to merge a certain dimension into the fact table given it is often used within *join* predicates, or to partition the fact table to speed up queries involving *scans*, or simply to precompute certain highly-frequent queries, and store their results as materialized views in a fast lookup data store. The method referred to in this first contribution deals precisely with the latter scenario. The rationale behind this workload-aware approach for view selection is to arrange expensive analytical queries into a consistent clustering configuration. Each of the resulting clusters contains highly similar queries, not only in terms of data attributes but also in the structure of the statements. Subsequently, the members within each cluster are combined into an all-encompassing query statement which gets materialized into a view in the data store. Upcoming (unseen) queries are then fitted into the clustering configuration and translated to be evaluated against the corresponding materialized view. When queries run against these precomputed views, query execution time is substantially reduced since these data structures are in general a small fraction of the size of the base dataset,

and the results of expensive *scan*, *join*, *sort* and *aggregate* operations are immediately available, as they have been calculated upfront.
The proposed method relies on the syntactic analysis of the query statements composing the workload at a given time. In this way, query statements get mapped to an equivalent vectorized representation, and subsequently, an agglomerative hierarchical clustering model is fitted against the collection of feature vectors that make up the analytical workload. Finally, the query groups derived from the clustering configuration are scored according to a custom consistency metric to decide which of them are worth materializing. This method was tested on two different setups using the widely adopted *Star Schema Benchmark* [1]. The first of these setups utilized a traditional *PostgreSQL* relational database while the second one was built on top of a *Hadoop* cluster using *Apache Spark* for query processing (See Appendix A). In both cases, the proposed method proved effective in identifying consistent groups of related queries and deriving a comprehensive set of materialized views out of said groups. Query performance was also evaluated on the views generated with this method. The results of this evaluation showed a substantial decrease in query response time of up to 99% in comparison to the processing time on the base dataset, with a storage footprint that added up to 13% of the size of the original data on disk.

6.2 Explora: A framework for enabling low-latency querying on live data streams

Typical visual exploration applications for geospatial time-series data present users with a dashboard composed of a number of charts and controls, and a two-dimensional map they can navigate through actions like panning or zooming. Users of these applications expect their actions to trigger immediate changes on the information the dashboard presents to them. Supporting such a responsive interaction on top of big and continuously growing spatio-temporal data remains a challenging task [2]. By identifying basic interaction patterns it is possible to build a data processing pipeline aimed at speeding up the queries that get triggered when instantiating said patterns. That is precisely the approach behind the Explora framework (**E**fficient e**XPLOR**ation through **A**ggregation) for supporting interactive querying on live spatiotemporal data streams. From the outset, Explora draws upon the user interaction patterns identified by Andrienko et al. [3] in which two main categories of exploratory actions are recognized: (*i*) *elementary tasks* aimed at obtaining a snapshot of the observed variable(s) at a certain point in time, and (*ii*) *general tasks* intended for examining

how the state of the variable(s) change over time across a particular spatial region. To serve the queries supporting these exploratory actions under low-latency constraints EXPLORA resorts to the continuous computation of data summaries over the incoming data stream, across discretized spatio-temporal bins of various resolutions. These data summaries are assembled into a *dynamic spatio-temporal raster-like* data structure optimized for fast retrieval. Queries are solved against these structures by combining the data summaries whose corresponding bins overlap with the query predicates in both spatial and temporal dimensions. Since the amount of summaries is comparatively smaller than the number of raw observations, the query processing time is expected to decrease significantly.
Two proof-of-concept implementations of EXPLORA were developed using different technology stacks, considering *smart cities* as a use-case scenario. The first implementation consists of a setup with a traditional PostgreSQL database with added support for time-series and geospatial data. The second setup relies on a distributed stream processing application implemented in *Apache Kafka*. Both setups count upon a Kafka message broker to handle the ingestion of data collected from non-stationary sensors in a smart city environment. A performance evaluation conducted on both implementations, using data from a smart city test-bed deployed in the city of Antwerp, Belgium [4], accounted for a substantial reduction in query response time. Complex queries supporting visual exploratory actions are sped-up up to two orders of magnitude in contrast to requests running on the raw time-series data. Since EXPLORA thrives on aggregation over a discretized space and time grid, this response time speedup entails a measurable degradation in query accuracy which amounts to less than 10%. These results proved the EXPLORA framework effective in serving sub-second latency querying over historical and live spatio-temporal data.

6.3 EXPLORA-LD: A platform for scalable publication of live summaries of spatio-temporal data

While the EXPLORA framework provides a mechanism for proactively computing data summaries as new data comes in to the system, the querying interface it defines for serving those synopsis structures does not quite favor reuse (caching) and pushes all the heavy lifting of query processing down to the server-side. This is particularly taxing for the system impacting its ability to cope with increasing load. EXPLORA-LD (*LD* standing for *Linked Data*) builds on top of the EXPLORA framework and promotes a more bal-

anced trade-off between server-side and client-side processing. To achieve this, the Explora-LD platform specifies a lightweight *Linked Data Fragments* (LDF) interface for publishing self-contained representations of each of the summaries being dynamically computed over the stream of spatio-temporal data. Said representations—labeled as *summary fragments*—are arranged in a spatio-temporal knowledge graph, which reflects the fragmentation strategy used to compute the encoded data summaries. These summary fragments are also endowed with hypermedia controls which enable clients (both users and automated agents) to traverse the knowledge graph. In that way, the responsibility of computing the complete answer to a given arbitrary query is delegated to the client, which proceeds by traversing the knowledge graph structure and requesting from the server the summary fragments needed to respond to the overall query. Besides offloading the server form complex processing, the proposed LDF interface entails two additional appealing properties, namely *extensive reuse*: since summary fragments are immutable resources, they can be indefinitely cached; and *incremental answering*: clients can display partial answers as they retrieve summary fragments from the server, which makes for an interactive and highly responsive user experience.

To estimate the benefits and limitations of the proposed approach, a benchmark evaluation was carried out to contrast the performance of the incremental LDF interface provided by Explora-LD against the original blocking querying interface from Explora in response to increasing load. Results indicate that Explora-LD is able to consistently provide instantaneous intermediate answers to user queries before the original Explora implementation is able to deliver the complete query response. Moreover, when running on cached summary fragments, query response time gets further reduced by around 50%. In these circumstances CPU usage measured at the proxy/cache nodes is largely negligible, while memory consumption remains mostly constant as the load increases. In this sense, Explora-LD provides a lightweight, cost-efficient alternative for delivering interactive querying on streams of spatio-temporal data, which additionally can scale with the system's load.

6.4 Explora-VR: An edge-assisted content prefetching method for serving multiple concurrent users

Just as the system load varies over time, data access patterns reflecting user interests are also highly prone to change. The fourth major contribution

detailed in this dissertation addresses precisely the challenge of adapting to dynamic access patterns in latency-sensitive applications with multiple concurrent users. Tile-based omnidirectional video streaming was chosen as a compelling use-case scenario, considering that data (i.e., video content) in this kind of services is defined in terms of several dimensions. These dimensions include space (video content gets fragmented into multiple spatial tiles), time (the video sequence is partitioned into sets of consecutive frames called *segments*), and quality (video tiles are available in several quality representations to adapt the content delivery to the prevailing network conditions). To tackle this challenge, this dissertation introduced EXPLORA-VR (*VR* standing for *virtual reality*), a network-assisted content prefetching method for tile-based 360° video streaming. EXPLORA-VR is grounded on the premise that multiple viewers in a near-live on-demand streaming setting would hold their focus on certain specific parts of the scene being presented to them via a head mounted device (HMD). On that premise, the proposed method aims at identifying those most salient video tiles in a per-video-segment basis. These sets of salient tiles, designated as *collective viewports*, are downloaded from the content server to a cache-enabled edge node located in close proximity to the viewers. In this way, video content that is likely to be requested by the active users thus far is made readily available through a low-latency link. EXPLORA-VR draws on viewport prediction techniques to anticipate the position of the user's *field of view* (FoV) for the subsequent video segments. Thanks to a stream processing pipeline, which takes inspiration from the EXPLORA framework, these individual predicted FoVs are combined into the above-mentioned collective viewports and arranged into an in-memory FIFO queue of limited size, which makes for a *collective playout buffer*. By consuming video content from this collective buffer hosted on a nearby edge server, clients perceive an increased network throughput, which drives them to request video tiles with higher quality representations. This in turn leads to an enhanced quality of experience (QoE) overall.

EXPLORA-VR was evaluated in an emulated environment using head movement traces collected from actual 360° video streaming sessions [5]. These traces comprise data gathered from 48 unique viewers, while watching nine different videos. Traces from three representative videos were used in the evaluation. The performance of the proposed content prefetching method was measured in terms of segment download time, and user QoE, namely delivered video quality, and occurrence of playback freezes. Results show that, under equivalent network conditions, EXPLORA-VR effectively outperforms the conventional *client-server* setup (i.e., without edge support), and a traditional caching implementation based on the least recently used

(LRU) replacement policy. In contrast to the LRU configuration, the setup using Explora-VR is able to deliver a freeze-free playback experience with higher video quality, as a result of a 40% increase in the perceived network throughput on the client's link (150% increase w.r.t. the conventional client-server setup). Moreover, Explora-VR can consistently deliver most of the clients' requests (>98%) from the cache-enabled edge server, which translates into a considerable reduction in backhaul traffic and content server load.

6.5 Future Perspectives

This dissertation presented several contributions in the domain of efficient data access methods for large multi-dimensional data collections. Of course, it would not be feasible to exhaust the subject matter within the limited scope of the research conducted, and therefore some challenges remain open, and promising venues for future studies have emerged as a result of the work documented herein. Below, these future research directions are briefly discussed.

Managing data integrity in big data systems

The mechanisms introduced in this dissertation intend to enable instant data access for latency-sensitive applications, thus preventing users and clients in general from taking action on stale data. However, in these approaches data integrity aspects—as pre-eminent as timeliness is in ensuring quality decision making—are largely taken for granted. The rise of *big data* has prompted a golden age for artificial intelligence (AI) and machine learning (ML) applications. In the modern digital world, these ML models fed on vast amounts of data have become pervasive, being increasingly deployed in several critical systems (e.g., self-driving vehicles, risk management and credit scoring, talent acquisition software, etc.). As the impact of the decisions we make based on these models grows, so does the need for ensuring the integrity and quality of the data they are trained on. In this sense, data traits such as correctness, completeness, consistency, and lineage are foundational for building trust in these big data systems. Explora-LD, the platform introduced in chapter 3, can be regarded as a step in this direction in that it adopts a metadata-rich, linked-data based representation of the data being served. The foregoing, added to the use of shared domain vocabularies to describe information resources, provide applications with valuable context as to where the data came from, and how it is expected to be used. There is still ample room for extending this research area including the

integration of context-aware mechanisms for anomaly detection [6] aiming to promote data integrity, and scalable methods for capturing data provenance in large datasets [7]. Likewise, semantic technologies can be used to augment ML models, enabling these systems to be not only accurate and reliable, but also predictable and explainable [8].

Impact of fault tolerance mechanisms

A large part of this dissertation deals with an extensive description of the EXPLORA framework and its derived implementations (i.e., EXPLORA-LD and EXPLORA-VR). These mechanisms for improving read performance heavily rely on continuous materialized views whereby they manage to anticipate and promptly answer clients' expensive queries, in a cost-efficient and scalable manner. One important aspect that is yet to be investigated in order to use the proposed methods in production-ready systems is the implementation of fault tolerance mechanisms. The proof-of-concepts built for evaluating the contributions in this dissertation assume sharded/partitioned data with a single replica per partition. This makes these systems highly vulnerable to outages of the datastores where continuous views are kept. Increasing the number of replicas can substantially reduce the impact of this kind of failures, boosting systems availability in consequence. However, the effective use and management of replicated data involve making a number of decisions, among them:

- how many replicas should handle write requests (i.e., whether to use *single-leader*, *multi-leader*, or *leaderless replication*),
- whether or not all replicas should acknowledge every write operation before it is considered successful (i.e., *synchronous* vs. *asynchronous replication*),
- the definition of quorum and the consensus mechanism for reaching agreement on the value of a replicated data element (e.g., protocols such as *Paxos* and *Raft* [9])

Each of the choices above involves varying trade-offs between system availability, data consistency and query responsiveness that are worth studying further and evaluating in practical evaluation settings.

Adaptive network-supported distributed query processing

The approach presented in chapter 5 features a method for edge-assisted query processing able to anticipate the information demands from multi-

ple clients in time-sensitive services. Besides a more in depth study of the effects of the variability of network conditions in the performance of the proposed method (e.g., fluctuating link capacity, packet loss, etc.), a further compelling line of research consists in investigating smart scheduling methods to intelligently scatter the processing workload across multiple edge nodes, including dynamic resource allocation mechanisms. In such a distributed setting, clients can be dynamically clustered together according to features such as shared access patterns, client's location, end client device, and quality of service (QoS) requirements. The intended scheduling mechanisms should respond to said clustering configuration and optimally allocate the limited computing resources of the edge network to serve the queries from each of the derived groups. Recent studies on *multi-access edge computing* (MEC) [10] such as those by Cai et al. [11] and Liang et al. [12] can provide valuable guidelines for the design and realization of these edge-assisted mechanisms for distributed query processing.

Explora-VR: Objective and subjective evaluation of QoE

Concerning the specific use case of immersive video streaming studied when defining the mechanisms behind Explora-VR, metrics such as the occurrence and duration of playback freezes, the video quality, and the perceived network throughput in the client's link were used as proxy to estimate the quality of experience (QoE) delivered to the viewer. To get a more accurate and reliable indication of the performance of the proposed approach in this regard, a more extensive evaluation of the QoE is recommended. Said evaluation should comprise both a subjective assessment—through measures such as the mean opinion score (MOS) provided by actual viewers—and an objective analysis which involves estimating indicators such as the video quality metric (VQM) and the structural similarity index measure (SSIM).

References

[1] Patrick E O'Neil, Elizabeth J O'Neil, and Xuedong Chen. The star schema benchmark (ssb), 2009. URL https://www.cs.umb.edu/~poneil/StarSchemaB.PDF. https://www.cs.umb.edu/~poneil/StarSchemaB.PDF. Last accessed: 2018-11-28.

[2] Chaowei Yang, Keith Clarke, Shashi Shekhar, and C. Vincent Tao. Big spatiotemporal data analytics: a research and innovation frontier. *International Journal of Geographical Information Science*, pages 1–14, 2019. doi: 10.1080/13658816.2019.1698743. URL https://doi.org/10.1080/13658816.2019.1698743.

[3] Natalia Andrienko, Gennady Andrienko, and Peter Gatalsky. Exploratory spatio-temporal visualization: an analytical review. *Journal of Visual Languages & Computing*, 14(6):503–541, 2003.

[4] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester. City of things: An integrated and multi-technology testbed for iot smart city experiments. In *2016 IEEE International Smart Cities Conference (ISC2)*, pages 1–8, Sep. 2016. doi: 10.1109/ISC2.2016.7580875.

[5] C. Wu, Z. Tan, Z. Wang, and S. Yang. A Dataset for Exploring User Behaviors in VR Spherical Video Streaming. In *Proceedings of the ACM Multimedia Systems Conference*, pages 193–198, 2017.

[6] Bhanukiran Vinzamuri, Elham Khabiri, Anuradha Bhamidipaty, Gregory Mckim, and Biren Gandhi. An end-to-end context aware anomaly detection system. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1689–1698. IEEE, 2020.

[7] Leslie F Sikos and Dean Philp. Provenance-aware knowledge representation: A survey of data models and contextualized knowledge graphs. *Data Science and Engineering*, 5(3):293–316, 2020.

[8] Gilles Vandewiele, Bram Steenwinckel, Filip De Turck, and Femke Ongenae. Mindwalc: mining interpretable, discriminative walks for classification of nodes in a knowledge graph. *BMC Medical Informatics and Decision Making*, 20(4):1–15, 2020.

[9] Heidi Howard and Richard Mortier. Paxos vs raft: Have we reached consensus on distributed consensus? In *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*, pages 1–9, 2020.

[10] Mahshid Mehrabi, Dongho You, Vincent Latzko, Hani Salah, Martin Reisslein, and Frank HP Fitzek. Device-enhanced mec: Multi-access edge computing (mec) aided by end device computation and caching: A survey. *IEEE Access*, 7:166079–166108, 2019.

[11] Zhipeng Cai and Tuo Shi. Distributed query processing in the edge assisted iot data monitoring system. *IEEE Internet of Things Journal*, 2020.

[12] Fan Liang, Wei Yu, Xing Liu, David Griffith, and Nada Golmie. Toward computing resource reservation scheduling in industrial internet of things. *IEEE Internet of Things Journal*, 8(10):8210–8222, 2020.

A

Automatic View Selection for Distributed Dimensional Data

⋆⋆⋆

L. Ordonez-Ante, G. Van Seghbroeck, T. Wauters, B. Volckaert, and F. De Turck.

Published in International Conference on Internet of Things, Big Data and Security (IoTBDS), May 2019.

Abstract Small-to-medium businesses are increasingly relying on big data platforms to run their analytical workloads in a cost-effective manner, instead of using conventional and costly data warehouse systems. However, the distributed nature of big data technologies makes it time-consuming to process typical analytical queries, especially those involving aggregate and join operations, preventing business users from performing efficient data exploration. In this sense, a workload-driven approach for automatic view selection was devised, aimed at speeding up analytical queries issued against distributed dimensional data. This paper presents a detailed description of the proposed approach, along with an extensive evaluation to test its feasibility. Experimental results shows that the conceived mechanism is able to automatically derive a limited but comprehensive set of views able to reduce query processing time by up to 89%–98%.

A.1 Introduction

Existing enterprise applications often separate business intelligence and data warehousing operations—mostly supported by *Online Analytical Processing* systems (OLAP)—from day-to-day transaction processing—a.k.a. *Online Transaction Processing* (OLTP) [1]. While OLTP systems rely mostly on write-optimized stores and highly normalized data models, OLAP technologies work on top of read-optimized schemas known as *dimensional data models*, which leverage on denormalization and data redundancy to run computationally-intensive queries typical from decision-support applications (e.g. reporting, dashboards, benchmarking, and data visualization), that would result in prohibitively expensive execution on fully-normalized databases.

Traditional data warehousing systems are expensive and remain largely inaccessible for most of the existing small-to-medium sized business (SMEs) [2]. However, thanks to the advent of big data, more and more cost-effective (often open-source) tools and technologies are made available for these organizations, enabling them to run analytical workloads on clusters of commodity hardware instead of costly data warehouse infrastructure. Yet in such a distributed setting, some of the common challenges of conventional data warehousing systems become even more daunting to deal with: the way data is scattered and replicated across distributed file systems such as Hadoop's HDFS [3] makes it computationally expensive and time-consuming to run *Aggregate-Select-Project-Join* (ASPJ) queries which are one of the foundational constructs of OLAP operations.

For typical data warehousing and related applications using materialized views is a common methodology for speeding up ASPJ-query execution. The associated overhead of implementing this methodology involves computational resources for creating and maintaining the views, and additional storage capacity for persisting them. In this sense, finding a fair compromise between the benefits and costs of this method is regarded by the research community as the *view selection problem.*

In this regard, this paper presents an automatic view selection mechanism based on syntactic analysis of common analytical workloads, and proves its effectiveness running on top of distributed dimensionally-modeled datasets. The paper explores the techniques devised for abstracting feature vectors from query statements, clustering related queries based on an estimation of their pairwise similarity, and deriving a limited set of materialized views able to answer the queries grouped under each cluster.

The remainder of this paper is organized as follows: Section A.2 addresses the related work. Section A.3 describes the *view selection problem* and

presents an overview of the proposed approach for tackling it. Section A.4 elaborates on the syntactic analysis conducted on analytical workloads, while Section A.5 describes a proof-of-concept implementation of the devised mechanism, along with the experimental setup and performance results. Finally conclusions and future work are addressed in Section A.6.

A.2 Related Work

Extensive research has been conducted around the view selection problem, as evidenced in several systematic reviews on the topic such as those by [4], [5], [6], [7].

The review elaborated in [6] groups existing approaches in three main categories: (*i*) heuristic approaches, (*ii*) randomized algorithmic approaches, and (*iii*) data mining approaches.

Heuristic and randomized algorithmic approaches emerged as an attempt to provide approximate optimal solutions to the NP-Hard problem that view selection entails. Both types of approaches use multidimensional lattice representations [8], AND-OR graphs [9, 10], or *multiple view processing plan* (MVPP) graphs [11, 12] for selecting views for materialization. Issues regarding the exponential growth of the lattice structure when the number of dimensions increases, and the expensive process of graph generation for large and complex query workloads, greatly impact the scalability of these approaches and their actual implementation in consequence [6, 13].

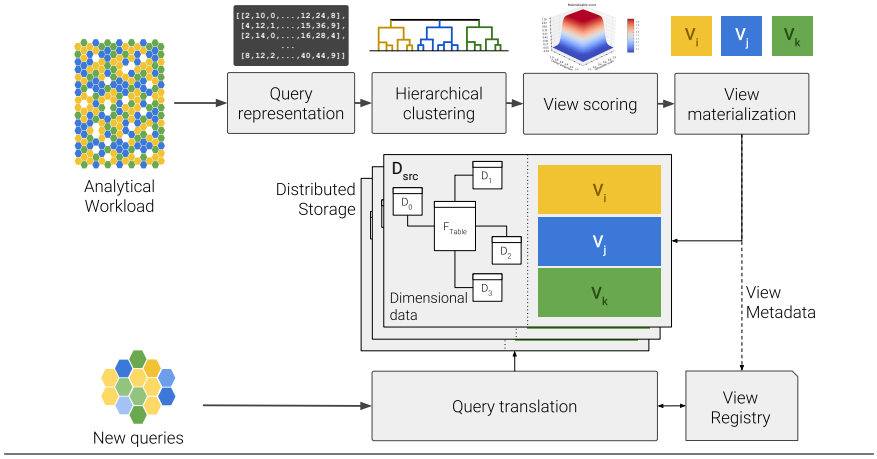
Unlike previously mentioned approaches, data-mining based solutions work with much simpler input data structures called *representative attribute matrices*, which are generated out of query workloads. These structures then configure a clustering context out of which candidate view definitions are derived. In [13, 14] candidate views are generated by merging views arranged in a lattice structure. Since the number of nodes in this lattice grows exponentially with the number of views, the procedure for traversing it can be expensive. Other data mining approaches for view selection, including the one from [15], involve browsing across several intermediate and/or historical results, which is deemed to be a very costly and unscalable process [6].

More recently, approaches such as [16] and [17] explore the application of materialized views on top of massive distributed data to speed up big data query processing. While the work of Goswami et al. [16] addresses a solution based on a multi-objective optimization formulation of the view selection problem, it assumes as given the set of candidate views from which the selection is made. On the other hand, [17] elaborates on the recently enabled

support for materialized views in Apache Hive [18], however at the time of writing there is no indication of any built-in mechanism for supporting view selection with this new feature.
In view of the above, this paper elaborates on an automatic mechanism for materialized view selection on top of distributed dimensionally-modeled data. The mechanism presented in the following sections relies on syntactic analysis of query workloads using a representative attribute matrix as input data structure, assembled as a collection of feature vectors encoding all the clauses of each individual query in the workload at hand. With this input, a strategy for selecting a limited set of candidate materializable views is implemented, comprising the use of hierarchical clustering along with a custom query distance function complying with the structure of the feature vectors, and the estimation of a *materializable score* on the resulting clustering configuration, allowing to unambiguously identify materializable groups of queries.

A.3 Materialized view selection

Figure A.1 Materialized view selection: architecture overview



Before addressing an overview of the proposed approach, let's first define the *view selection problem.*

Definition A.1 View selection problem. *Based on the definition by Chirkova et al. [19]: Let $\mathcal{R}$ be the set of base relations (comprising fact(s) and dimensions tables), $\mathcal{S}$ the available storage space, $\mathcal{Q}$ a workload on $\mathcal{R}$, $\mathcal{L}$ the function for estimating the cost of query processing. The view selection*

problem is to find the set of views $\mathcal{V}$ (view configuration) over $\mathcal{R}$ whose total size is at most $\mathcal{S}$ and that minimizes $\mathcal{L}(\mathcal{R}, \mathcal{V}, \mathcal{Q})$

In the context of the view selection approach proposed herein, some assumptions are made for the system to identify and materialize candidate views out of the syntactic analysis of query workloads:

1. The source data collection (D_{src}) complies a star schema data model, i.e. it comprises a fact table referencing one or more dimension tables.
2. D_{src} is *temporary immutable*. This is a common scenario in some data warehousing systems, where analytical data is updated once in a substantial period of time —e.g. through an ETL procedure running on top of OLTP databases—, and queried multiple times during such a period.
3. Statistical information regarding D_{src}, such as the size (row count) of each of the base tables composing the dataset, as well as the cardinality of the attributes that make up these relations is available either by querying the metadata kept by the datastore, or by directly querying the base tables.
4. Latency is favored over view storage cost. This means that the decision on materializing candidate views is driven not by storage restrictions, but by the gain in query latency.

Figure A.1 outlines the main components of the mechanism proposed herein to address the stated view selection problem. In terms of the definition A.1, given a dimensionally modeled dataset $\mathcal{R}$ and a workload $\mathcal{Q}$, the view selection mechanism starts by translating the queries in $\mathcal{Q}$ into feature vectors representing the attributes contained in each of the clauses of an ASPJ-query, i.e. aggregate operation, projection, join predicates and range predicates. In contrast to similar query representations such as the one used in [13], the method proposed herein accounts not only for query-attribute usage, but also for query structure by defining a number of regions/segments representative of each of the clauses of a Select-Project-Join (SPJ) query, i.e. aggregate operation, select list, join predicates and range predicates. This way, the devised query representation provides a more precise specification of the query statements in $\mathcal{Q}$.
The collection of feature vectors of $\mathcal{Q}$ configure a clustering context $\mathcal{C}$. This context is then fed to a clustering algorithm able to identify groups of related queries based on a similarity score computed via a custom query distance function. Upon running the clustering job, the resulting clustering configuration $\mathcal{K}$ comprises several groups of queries the algorithm deemed to be

similar. The idea behind building this clustering configuration is to be able to deduce view definitions covering the queries arranged under each cluster. The clustering algorithm might come up with spurious clusters, i.e. groups of queries that are actually not that related. To identify those spurious clusters and setting them apart from those clusters whose corresponding candidate views are worth materializing, a *materializable score* is defined, taking into account a measure of cluster consistency and the cluster size. Further details on this score and the clustering procedure are provided later in section A.4.2.
Based on the results of the *materializable score* computed on the clustering configuration $\mathcal{K}$, a subset of the candidate views in $\mathcal{V}$, $\mathcal{V}_{mat}$, is prescribed to be materialized. Finally, with the views in place, the translation of new analytical queries matching said views is performed.

A.4 Query Analysis

The syntactic analysis this work thrives on, starts by mining the information contained in the *select list* and *search conditions* clauses, encoding these values in a feature vector representation that enables further query processing.

A.4.1 Query representation

The procedure for obtaining a text-mining-friendly representation of the queries takes each one of the `SELECT` statements from a workload $\mathcal{Q}$ and extracts the
aggregate (ag_q) and *projection* (pj_q) elements, and *join* (jn_q) and *range* (rg_q) predicates, resulting in the following tuple:

$$q = (ag_q, pj_q, jn_q, rg_q) \tag{A.1}$$

The tuple above is the high-level vector representation of the queries from $\mathcal{Q}$. Consider for example the following `SELECT` statement:

```
SELECT SUM(lo_revenue), d_year, p_category
FROM lineorder, dwdate, part
WHERE lo_orderdate = d_datekey
    AND lo_partkey = p_partkey
    AND d_year > 2010
GROUP BY d_year, p_category
```

For the query above:

$$ag_q = [\texttt{SUM}, \texttt{lo\_revenue}]$$

$$pj_q = [\texttt{d\_year}, \texttt{p\_category}]$$

$$jn_q = [\texttt{d\_datekey}, \texttt{p\_partkey}]$$

$$rg_q = [\texttt{d\_year}]$$

Each element of the above high-level vector representation gets mapped to a vector using a binary encoding function, as described below.

Definition A.2 Binary mapping function. *Let* R *be a relation defined as a set of* m *attributes* $(a_1, a_2, ..., a_\mathsf{m})$ *—with* a_m *being the primary key of* R*—, and given* r *an arbitrary set of attributes, the binary mapping of* r *according to* R*, denoted by* $bm_R(\mathsf{r})$*, is defined as follows:*

$$bm_\mathsf{R}(\mathsf{r}) = \{b_i\}, 1 \leq i \leq \mathsf{m}$$
$$b_i = \begin{cases} 1, & \textit{if } a_i \in \mathsf{r} \\ 0, & \textit{otherwise} \end{cases} \tag{A.2}$$

Using the mapping function above, the vector representation of each one of the query elements in Eq.A.1 (designated henceforth as *segments*), for a dimensional schema comprising one fact table and N dimension tables, is defined as follows:

$$\mathbf{ag_q} = [aggOpCode, bm_{\mathsf{Fact}}(ag_q)]$$

$$\mathbf{pj_q} = [bm_{\mathsf{Fact}}(pj_q), bm_{\mathsf{Dim1}}(pj_q), ..., bm_{\mathsf{DimN}}(pj_q)]$$

$$\mathbf{jn_q} = [bm_{\mathsf{Dim1}}(jn_q), ..., bm_{\mathsf{DimN}}(jn_q)]$$

$$\mathbf{rg_q} = [bm_{\mathsf{Fact}}(rg_q), bm_{\mathsf{Dim1}}(rg_q), ..., bm_{\mathsf{DimN}}(rg_q)]$$

where *aggOpCode* designates the aggregate operation using one-hot encoding, namely, `COUNT`: 00001, `SUM`: 00010, `AVG`: 00100, `MAX`: 01000, `MIN`: 10000.
A complete feature vector $\mathbf{q}$ representing a query $q \in \mathcal{Q}$ is set by putting together the above-mentioned segments, that is:

$$\mathbf{q} = [\mathbf{ag_q}, \mathbf{pj_q}, \mathbf{jn_q}, \mathbf{rg_q}]$$

Accordingly, considering the `SELECT` statement in the example above and the dimensional schema described in [20] which comprises one fact table and four dimension tables, a complete feature vector instance (its decimal equivalent for length and clarity) is shown below:

$$\mathbf{q} = [[2, 8], [0, 0, 16, 8, 0], [0, 1, 1, 0], [0, 0, 16, 0, 0]]$$

The collection of feature vectors representing the queries from $\mathcal{Q}$ are arranged as a *representative attribute matrix*, configuring a clustering context $\mathcal{C}$.

A.4.2 Query clustering and View materialization

Algorithm A.1 WPGMA clustering procedure

1: $\mathcal{K} \leftarrow \mathcal{C}; \mathcal{C} = \{\mathbf{q_0}, \mathbf{q_1}, \ldots, \mathbf{q_N}\}$ ▷ Initializing clusters (singleton clusters)
2: $\mathbf{D} \leftarrow qDst(\mathbf{q}_i, \mathbf{q}_j)$ for all $\mathbf{q}_i, \mathbf{q}_j \in \mathcal{K}, i \neq j$ ▷ Pairwise dissimilarity matrix
3: $\mathbf{L} \leftarrow []$ ▷ Output matrix
4: **while** $|\mathcal{K}| > 1$ **do**
5: $(\mathbf{a}, \mathbf{b}) \leftarrow argmin(\mathbf{D})$ ▷ Get the nearest clusters
6: append $[\mathbf{a}, \mathbf{b}, \mathbf{D}[\mathbf{a}, \mathbf{b}]]$ to $\mathbf{L}$
7: remove $\mathbf{a}$ and $\mathbf{b}$ from $\mathcal{K}$
8: create new cluster $\mathbf{k} \leftarrow \mathbf{a} \cup \mathbf{b}$ ▷ Merge $\mathbf{a}$ and $\mathbf{b}$ into one cluster
9: update $\mathbf{D}$: $\mathbf{D}[\mathbf{k}, \mathbf{x}] = \mathbf{D}[\mathbf{x}, \mathbf{k}] = \frac{qDst(\mathbf{a},\mathbf{x}) + qDst(\mathbf{b},\mathbf{x})}{2}$ for all $\mathbf{x} \in \mathcal{K}$
10: $\mathcal{K} \leftarrow \mathcal{K} \cup \mathbf{k}$
11: **end while**
12: **return** $\mathcal{K}, \mathbf{L}$ ▷ $\mathbf{L}$: WPGMA dendrogram: $((N-1) \times 3)$-matrix

The view selection approach documented herein relies on *hierarchical clustering* [21] for deriving groupings of similar queries. In contrast to other well-known clustering methods such as *K-Means* or *K-medoids*, hierarchical clustering analysis does not require the number of clusters upfront as parameter. Instead, it generates a hierarchical representation of the entire clustering context in which observations and groups of observations are stacked together from lower to higher levels, according to a distance measure based on the pairwise dissimilarities among the observations.

This way, a *dissimilarity metric* is required to apply hierarchical clustering analysis on a clustering context $\mathcal{C}$, along with a *linkage criterion* which estimates the dissimilarity among groups of queries as a function of the pairwise distance computed between queries belonging to those groups. In this sense, a distance function, $qDst$, is defined in which similarity between two queries is determined to be proportional to the number of attributes they share in a per-segment and per-relation (fact and dimensions) basis. Since vectors in $\mathcal{C}$ do not lie in an euclidean space, the *Weighted Pair Group Method with Arithmetic Mean* (WPGMA) clustering method is used as linkage criterion, instead of methods such as *centroid*, *median*, or *ward* [22].

Under this set-up, the clustering procedure (detailed in algorithm A.1) starts by assigning each query to its own cluster (see line 1). Then, the pairwise

dissimilarity matrix between these singleton clusters, $\mathbf{D}$, is computed and an empty matrix ($\mathbf{L}$) specifying the resulting dendrogram is initialized (lines 2-3). From $\mathbf{D}$, the two most similar (nearest) clusters are merged into one, and appended to $\mathbf{L}$ along with the distance between them (lines 5-6). Then, the pairwise dissimilarity matrix gets updated using the WPGMA method for computing the distance between the newly formed cluster and the rest of the currently existing clusters (eq. A.3):

$$\mathbf{D}[(\mathbf{a} \cup \mathbf{b}), \mathbf{x}] = \frac{qDst(\mathbf{a}, \mathbf{x}) + qDst(\mathbf{b}, \mathbf{x})}{2}, \quad (\mathbf{a}, \mathbf{b} \quad and \quad \mathbf{x} \quad being \quad clusters) \tag{A.3}$$

This procedure is then repeated until there is only one cluster left. Finally, both the clustering configuration ($\mathcal{K}$) and the dendrogram matrix ($\mathbf{L}$) are returned (line 12).
As mentioned earlier, spurious clusters might be found in the derived clustering configuration. To avoid further processing of those query groups a score was defined indicating to what extent it is worth to materialize the view derived from a particular cluster.

Definition A.3 Materializable cluster. *A cluster* $\mathbf{c}$ *from a clustering configuration* $\mathcal{K}$ *is said to be* materializable *if the following conditions are met:*

1. *Queries in* $\mathbf{c}$ *are highly similar to each other.*
2. *Queries in* $\mathbf{c}$ *are clearly separated (highly dissimilar) from queries in other clusters.*
3. $|\mathbf{c}|$ *is large enough in proportion to the size of the workload* $|\mathcal{Q}|$.

A cluster meeting the first two conditions is said to be a *consistent cluster*, while the third condition prevents singleton and small clusters from being further processed. Based on the above definition, the *materializable score* of a cluster ($mat(\mathbf{c})$ in eq. A.4) is computed as the product of two sigmoid functions: one on the per-cluster *silhouette score* (S) [23]—defined below in eq. A.5—and the other on the per-cluster proportions (P).

$$mat(\mathbf{c}) = \Big(\frac{1}{1 + e^{-k(S(\mathbf{c}) - s_0)}}\Big)\Big(\frac{1}{1 + e^{-k(P(\mathbf{c}) - p_0)}}\Big) \tag{A.4}$$

With:

$$S(\mathbf{c}) = \frac{1}{|\mathbf{c}|} \sum_{\mathbf{q}_i \in \mathbf{c}} \frac{b(\mathbf{q}_i) - a(\mathbf{q}_i)}{max\left\{a(\mathbf{q}_i), b(\mathbf{q}_i)\right\}}, \quad P(\mathbf{c}) = \frac{|\mathbf{c}|}{|\mathcal{Q}|} \tag{A.5}$$

Where,

- k is a factor that controls the steepness of both of the sigmoid functions,
- s_0 and p_0 are the midpoints of the silhouette and cluster-proportion sigmoids respectively,
- $a(\mathbf{q}_i)$ is the average distance between $\mathbf{q}_i$ and all queries within the same cluster,
- $b(\mathbf{q}_i)$ is the lowest average distance of $\mathbf{q}_i$ to all queries in any other clusters.

Upon factoring out the spurious clusters, the next step is deriving view definitions covering the queries arranged under each of the *materializable clusters* ($\mathcal{K}_{mat} \subseteq \mathcal{K}$). Algorithm A.2 below details the procedure conducted to derive the views V_i meeting this containment condition on each of the materializable clusters. In this procedure, the ASPJ clauses of the resulting views are defined in terms of the union of the corresponding attributes from each query in the cluster (*aggregate* ($ag_{\mathbf{q}}$), *projection* ($pj_{\mathbf{q}}$), *join* ($jn_{\mathbf{q}}$), and *range* ($rg_{\mathbf{q}}$) predicates in lines 4-7).

Algorithm A.2 Procedure for deriving view definitions

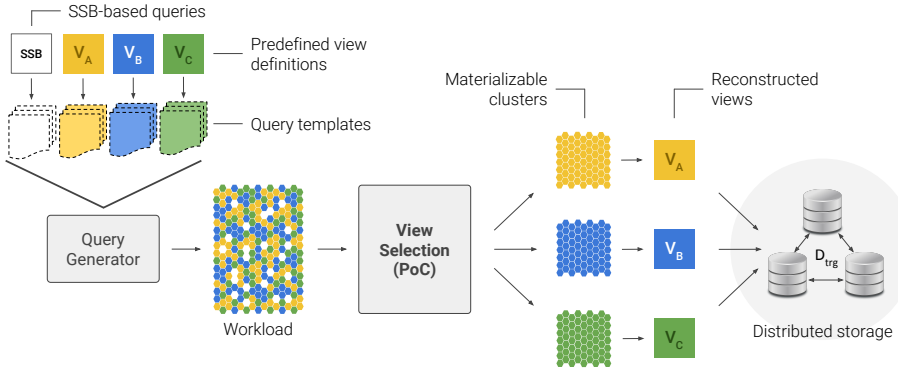
1: Let $\mathbf{c}$ be a cluster in $\mathcal{K}_{mat}$
2: $V \leftarrow [ag_V, pj_V, jn_V, groupBy_V]$ ▷ Output view definition
3: **for** each query $\mathbf{q}$ in $\mathbf{c}$ **do**
4: $\quad ag_V \leftarrow ag_V \cup ag_{\mathbf{q}}$
5: $\quad pj_V \leftarrow pj_V \cup pj_{\mathbf{q}} \cup rg_{\mathbf{q}}$
6: $\quad jn_V \leftarrow jn_V \cup jn_{\mathbf{q}}$
7: $\quad groupBy_V \leftarrow groupBy_V \cup pj_{\mathbf{q}} \cup rg_{\mathbf{q}}$
8: **end for**
9: **return** V

A.5 Evaluation

A.5.1 Proof-of-concept Implementation

A bottom-up approach was adopted to test the view selection mechanism detailed in the previous sections. In this way, starting from a set of predefined

Figure A.2 Proof-of-concept implementation of the proposed view selection mechanism.



view definitions, the effectiveness of the proposed mechanism is estimated in terms of its ability for identifying the same set of views and reconstructing their definitions, upon analyzing a query workload generated from query templates fitting the original set of views (see Figure A.2).

This proof-of-concept implementation leverages the *Star Schema Benchmark* (SSB) as baseline schema and dataset, and therefore both the predefined views and query templates, as well as the query generator module were designed and built so they conform to the data model the SSB embodies. Thirteen ASPJ-query statements compose the full query set of the SSB, arranged in four categories/families designated as *Query Flights* (a detailed definition of the SSB is available at [20]). For this proof-of-concept, three view definitions were derived based on the original SSB query set, and from each view definition, four query templates were prepared. Additionally, one template per each one of the 13 canonical SSB queries were also composed. With this set of 25 templates as input, a module that generates random instances of runnable queries enabled the creation of query workloads of arbitrary size. Listings below present the definitions of each one of the mentioned views.

```
SELECT sum(lo_revenue), p_brand1, c_region,
       s_region, d_year
FROM lineorder, customer, dwdate, part, supplier
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY p_brand1, c_region, s_region, d_year
ORDER BY p_brand1, c_region, s_region, d_year
```

Listing A.1: Definition of View A

```
SELECT sum(lo_ordtotalprice), p_category, c_city,
       s_city, d_yearmonthnum
FROM lineorder, customer, dwdate, part, supplier
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY p_category, c_city, s_city, d_yearmonthnum
ORDER BY p_category, c_city, s_city, d_yearmonthnum
```

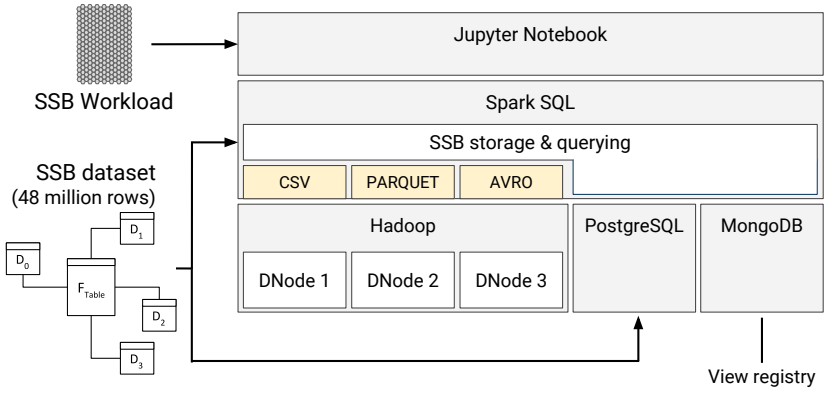
Listing A.2: Definition of View B

```
SELECT sum(lo_supplycost - lo_tax), c_region, p_mfgr,
       s_region, c_nation, d_year
FROM lineorder, customer, dwdate, part, supplier
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY c_region, p_mfgr, s_region, c_nation, d_year
ORDER BY c_region, p_mfgr, s_region, c_nation, d_year
```

Listing A.3: Definition of View C

A.5.2 Definition of the data serialization format

Figure A.3 Set-up for deciding on the data serialization format

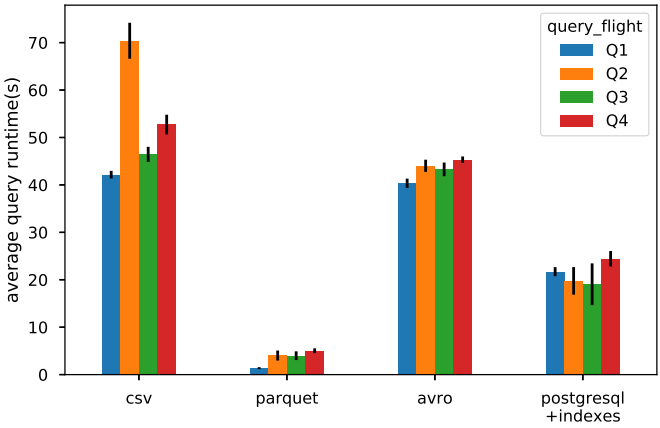


Since serialization formats determine the way data structures are turned into bytes and sent over the network, and how said structures are stored on disk, such formats have a major impact on the response time of data processing and retrieval operations performed in a distributed fashion. This is why, prior to evaluating the performance of the proposed view selection mechanism, the decision on which serialization format to use for encoding and storing the SSB datasets into HDFS needed to be made. Figure A.3

outlines the setup arranged to conduct a benchmark analysis on three different data serialization formats, one being a text-based format (`CSV`) and two binary schema-driven formats (`Parquet` [24] and `Avro` [25]). By leveraging on the built-in support Spark SQL provides for these serialization formats, a 48-million-row SSB dataset was encoded into CSV, Parquet and Avro and stored in HDFS. Then, the canonical SSB query set (consisting of 13 queries) was run against each of the encoded datasets, as well as against a separate dataset placed in a single-node PostgreSQL[1] database serving as a reference.

Figure A.4 shows the results obtained from measuring the average query runtime (over 10 runs) for each one of the serialization formats. Notice how, in the mentioned conditions, queries running against the Parquet-encoded dataset ran up to 10 times faster than the reference relational database. Also, Parquet was the only serialization format that managed to outperform the average query runtime of PostgreSQL.

Figure A.4 Average query-flight runtime per data serialization format ($SSB\ SF = 8$)

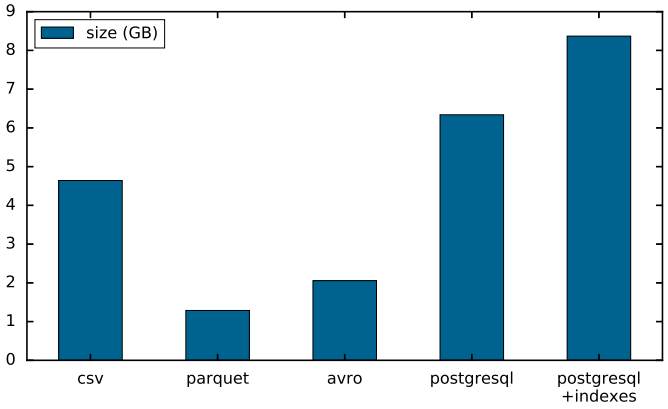


Serialization formats also have a significant impact on the size of the encoded data structures. While for text-based human-readable formats such as CSV data is stored *as-is*, binary formats like Parquet and Avro do apply compression on the data they encode. Figure A.5 shows a comparison between the reported sizes (in gigabytes) of the SSB dataset for each of the considered serialization formats. According to these results Parquet is once again the most efficient serialization format, reaching a compression

[1]PostgreSQL 9.5.8 working with the default configuration and deployed on a VMWare® virtual machine as the ones used for the Hadoop cluster (postgresql.org)

ratio of 3.6:1 in relation to the uncompressed CSV-encoded dataset, and 6.5:1 in relation to the PostgreSQL reference database (including primary key indexes). In consequence, Parquet was the chosen serialization format for encoding the various datasets involved in the evaluation of the proposed view selection approach.

Figure A.5 Disk space usage per data serialization format ($SSB\ SF = 8$)



A.5.3 Experimental setup

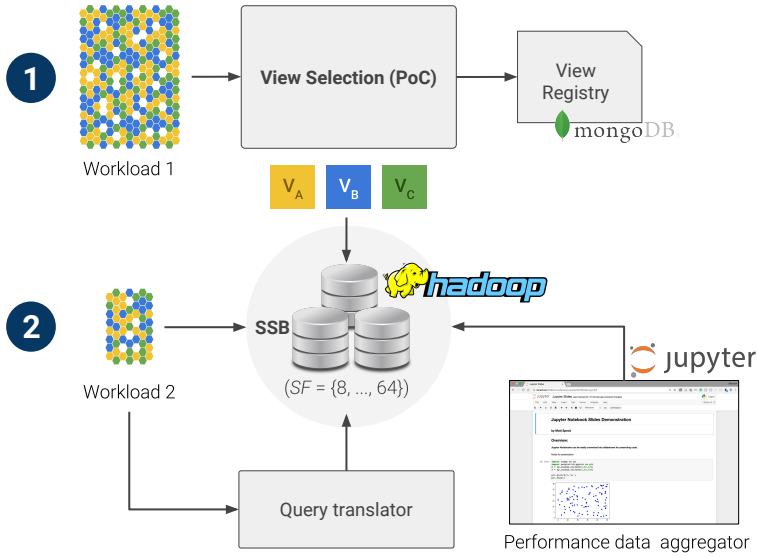
Figure A.6 depicts the arrangement of components and technologies used for conducting the experimental evaluation of the proposed view selection approach. This evaluation comprised two major stages:

(*1*) Running the view selection implementation on a 400-query workload to the point it materializes views A, B, and C (defined in section A.5.1), while keeping track of the runtime involved in the procedures of *query clustering*, *view scoring* (using the materializable score defined in section A.4.2), and *view creation (i.e. materialization)*.

(*2*) Once the views are materialized, run a 100-query workload against both the base SSB dataset and the materialized views. In doing the latter, workload queries first pass through a translation component that gathers the details of the available materialized views from the *view registry* (stored in a *MongoBD*[2] 2.6.10 document database), and adapts the incoming query statements accordingly.

[2]Available at mongodb.com

Figure A.6 View selection experiment set-up



For all the stages, the performance information collected from running the tests were aggregated and visualized using *Jupyter notebook*[3]. During this evaluation, workloads were run against eight different sizes of the SSB dataset, as specified below in table A.1. These datasets were stored into a 3-Node *Hadoop*[4] 2.7.3 cluster deployed on 3 VMWare® virtual machines, each one with the following specifications: Intel® Xeon® E5645 @2.40GHz CPU, 16GB RAM, 250GB hard disk.

Table A.1: SSB dataset sizes

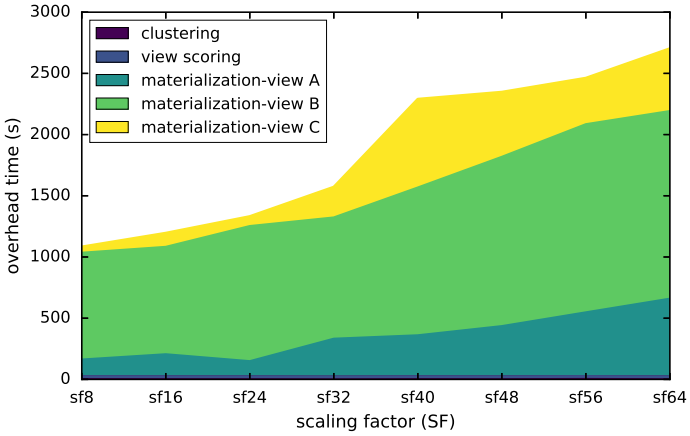
Scaling Factor (SF)	**Dataset size (# rows)**
8	48×10^6
16	96×10^6
24	144×10^6
32	192×10^6
40	240×10^6
48	288×10^6
56	336×10^6
64	384×10^6

[3]Available at jupyter.org
[4]Available at hadoop.apache.org

A.5.4 Results

A.5.4.1 View selection overhead

Figure A.7 View selection runtime per process ($|\mathcal{Q}| = 400$). Time needed for view materialization grows as the dataset size increases, while the overhead due to clustering and view scoring remains invariant.

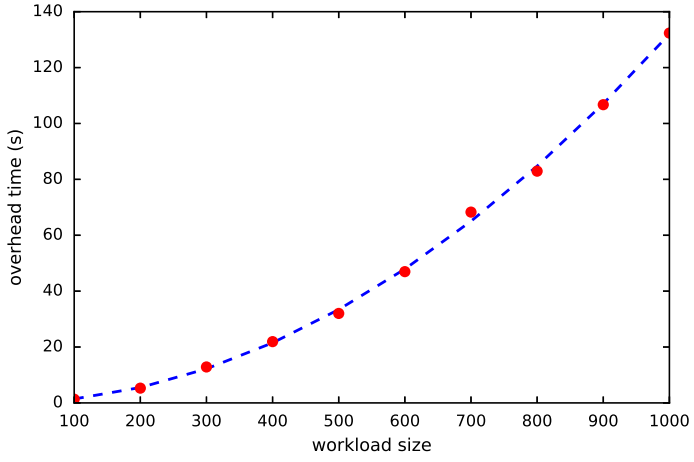


The overhead of the view selection implementation was estimated first by running it on a 400-query workload throughout the considered range of sizes of the SSB dataset. Results show that, for a fixed-size workload, the runtime overhead grows nearly proportional to the size of the data collection (See Figure A.7), with a major part of said overhead due to the view materialization itself. As mentioned before, the proposed view selection mechanism involves the execution of a sequence of steps: (1) *query clustering*, (2) *view (or cluster) scoring*, and (3) *view materialization*. Out of these only the first two steps have to do with the syntactical analysis of query sets described throughout this paper, while the last one refers to the actual materialization of the derived views in Parquet. Figure A.7 shows the execution times for each one of the three mentioned steps, including the individual materialization of each one of the three selected views. Note how clustering and view scoring amount to only 20 seconds, and remain largely invariant as the dataset size grows larger. Nonetheless, it is worth mentioning that the behaviour evidenced in Figure A.7 for the materialization step cannot be assumed the same for any arbitrary set of views, since this part of the runtime overhead depends not only on the size of the dataset, but also on factors

such as view size, the join predicates in the view definition, and —given that views are placed in a distributed file system— also the latency of the network.

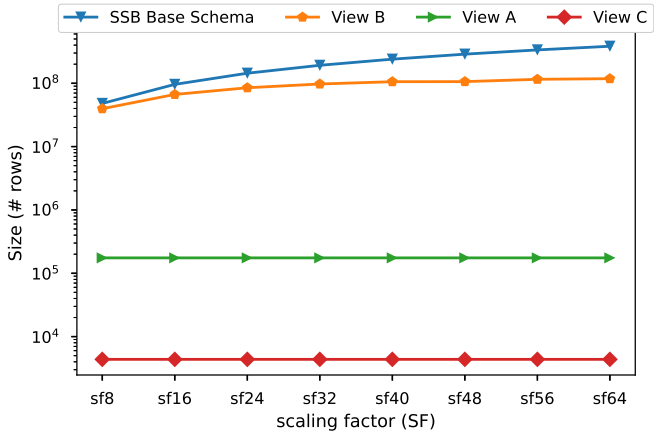
On the other hand, the implementation of the WPGMA method used in the clustering analysis relies on the *nearest-neighbors chain algorithm* which is known to have $O(N^2)$ time complexity [22]. As Figure A.8 shows, the overhead due to said analysis features a quadratic growth as the number of queries in the workload increases, outperforming alternative approaches with exponential complexity discussed back in section A.2.

Figure A.8 View selection overhead vs Workload size. The syntactical analysis features a quadratic growth w.r.t. the size of the query set.



When it comes to storage cost, view size varies depending on the cardinality of the fields used in the group-by clause of their definition [13], and whether or not there are hierarchical relations between such attributes (e.g. the one between `c_region`, `c_nation` and `c_city`). Figure A.9 shows the size per materialized view, as well as the number of rows in the SSB base schema for a range of values of the scaling factor (SF). Notice that while the number of records of views A and C is fairly negligible in comparison with the base schema, view B and the base dataset have comparable sizes for small values of SF. Then, the size of view B tends to stabilize around 10^8 records as the base data set gets larger. Such difference in size among the views has to do with the cardinality of the fields used when defining said views. For instance, view B uses fields `c_city`, `s_city` and `d_yearmonthnum` whose cardinality is far larger than fields such as `c_region`, `c_nation` and `d_year`, used in the remaining views.

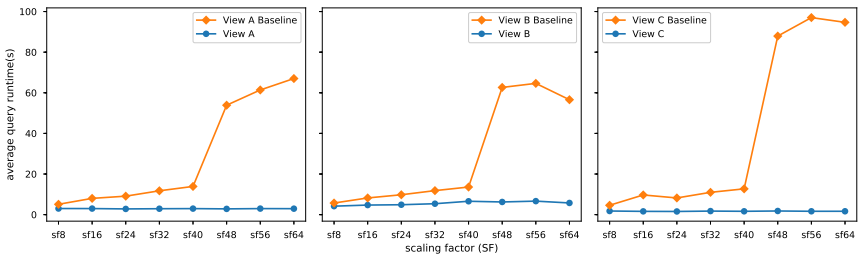
Figure A.9 View size overhead vs SSB scaling factor



A.5.4.2 View selection performance

With the selected views already materialized, a 100-query workload was run against each one of the eight base SSB datasets to get a query latency baseline. Out of those 100 queries, 75 were covered by the three available materialized views (25 queries per view), and the remaining ones were canonical SSB-based queries. Once the latency baseline was built, the same workload was issued this time with the query translation module in place, so that incoming queries matching any of the definitions of the available materialized views get rewritten and issued against them. Figure A.10 illustrates the contrast between the baseline query runtime and the response time when queries run against materialized views. In the light of these results it is worth to highlight three key facts:

Figure A.10 Query runtime per view: Baseline runtime vs. View runtime ($|\mathcal{Q}| = 100$, Spark running on *yarn-client* mode with 3 executors and $spark.executor.memory = 1Gb$)



(*i*) For all the selected views the time required for queries to run against the base dataset steadily grows as the scaling factor (SF) increases from 8 (48 million rows) to 40 (240 million rows). This describes an expected behaviour since the base dataset is also growing at a uniform rate (48 million rows per step).

(*ii*) From $SF = 48$ (288 million rows) onwards, there is a strong though less regular increase in the average response time of queries issued against the base datasets: queries run 5-9 times slower compared to those running on the immediate smaller dataset ($SF = 40$), when only a 22-27% mean increase in the query runtime was expected. Such a stark difference is consequence of Apache Spark changing the mechanism it uses to implement join operations. Up to $SF = 40$, the size of the dimension tables is small enough for Spark to broadcast them across the executors —collocated with the Hadoop DataNodes— and use its most performant join strategy known as *Broadcast Hash Join*. However, for $SF \geq 48$ some of those dimensions grows larger than the threshold set in Spark for them to be regarded as *broadcastable* datasets, compelling Spark to fall back to *Sort-Merge Join*, which entails an expensive sorting step on the tables involved in the join operation, ultimately impacting the query response time.

(*iii*) Materialized views outperform the base datasets for any value of $SF \geq 8$. Queries running on views A and C perform 2-8 times faster than the corresponding base dataset for values of SF between 8 and 40, and 22-65 times faster for larger values of SF. Likewise, queries running against the second view (view B) run up to 2 times faster than queries running on the base dataset for $SF \leq 40$, and up to 10 times faster for larger values of SF. The expensive join operations performed for queries running on the base dataset are bypassed for those matching any of the available selected views, allowing Spark to run those queries in a fraction of their original query response time.

Table A.2 summarizes the results obtained from running the above test, stating the reduction in query runtime achieved through each one of the views relative to the average baseline query runtime.

A.6 Discussion and Conclusions

The analytical workloads typical in OLAP applications feature expensive data processing operations whose cost and complexity increases when running on a distributed setting. By identifying recurrent operations in the

Table A.2: Query latency reduction per view ($|\mathcal{Q}| = 100$)

	% Reduction in query response time		
SSB Dataset size (SF)	**View A**	**View B**	**View C**
8	40.86	26.99	61.21
16	62.66	42.64	83.36
24	69.05	50.14	80.95
32	75.16	54.48	84.06
40	78.60	51.56	87.05
48	96.02	88.96	98.00
56	96.10	89.15	98.24
64	95.36	89.82	98.46

queries composing said workloads and saving their resulting output on disk or memory in the form of distributed views, it is possible to speed up the processing time of not only known but also previously unseen queries. That is precisely the premise behind the mechanism detailed in this paper, which leverages on syntactic analysis of OLAP workloads for identifying groups of related queries and deriving a limited but comprehensive set of views out of them. The views the devised mechanism comes up with proved an effective method for circumventing expensive distributed join operations and subsequently reducing the query processing time by up to 89%–98% with reference to the runtime on the base distributed dimensional data.

While the convenience of distributed materialized views is more prominently perceived as the dimensional data grows larger, one of the main open challenges of the proposed approach has to do with the unbounded size of the views that the mechanism is able to compose, which increases the associated processing overhead and cuts down the relative benefit of using these redundant data structures. To cope with this limitation, the view selection mechanism needs to be aware not only of the recurrent attributes and operations of queries but also of the cardinality of such attributes, so that views including attributes with high cardinality (consider for instance view B in section A.5.1) get materialized as multiple size-bounded child views corresponding to partitions of the original view. Additionally, by keeping track of how the selection conditions of incoming workloads change over time, it is possible to implement a continuous view maintenance strategy that performs horizontal partitioning on the derived views, allowing the proposed mechanism to adapt to workload-specific demands, using an approach similar to the one presented in [26]. The implementation of the cardinality-

awareness feature for the proposed view selection mechanism, as well as the view maintenance strategy discussed above are the future extensions of the work presented in this paper.

References

[1] Hasso Plattner. *A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases.* Springer Publishing Company, Inc., 2013. ISBN 364236523X, 9783642365232.

[2] U. B. Qushem, A. M. Zeki, A. Abubakar, and S. Akleylek. The trend of business intelligence adoption and maturity. In *UBMK 2017*, pages 532–537, Oct 2017. doi: 10.1109/UBMK.2017.8093455.

[3] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *MSST 2010*, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-1-4244-7152-2. doi: 10.1109/MSST.2010.5496972. URL http://dx.doi.org/10.1109/MSST.2010.5496972.

[4] Garima Thakur and Anjana Gosain. A comprehensive analysis of materialized views in a data warehouse environment. *International Journal of Advanced Computer Science and Applications*, 2(5):76–82, 2011. ISSN 2156-5570. doi: 10.14569/IJACSA.2011.020513.

[5] T Nalini, A Kumaravel, and K Rangarajan. A comparative study analysis of materialized view for selection cost. *World Applied Sciences Journal (WASJ)*, 20(4):496–501, 2012.

[6] Rajib Goswami, Dhruba Kr Bhattacharyya, Malayananda Dutta, and Jugal K. Kalita. Approaches and issues in view selection for materialising in data warehouse. *International Journal of Business Information Systems*, 21(1):17–47, December 2016. ISSN 1746-0972. doi: 10.1504/IJBIS.2016.073379.

[7] Anjana Gosain and Kavita Sachdeva. A systematic review on materialized view selection. In Suresh Chandra Satapathy, Vikrant Bhateja, Siba K. Udgata, and Prasant Kumar Pattnaik, editors, *FICTA 2017*, pages 663–671, Singapore, 2017. Springer Singapore. ISBN 978-981-10-3153-3.

[8] Maria Trinidad Serna-Encinas and Jose Antonio Hoyo-Montano. Algorithm for selection of materialized views: based on a costs model. In

Current Trends in Computer Science, 2007. ENC 2007. Eighth Mexican International Conference on, pages 18–24. IEEE, 2007.

[9] Xia Sun and Ziqiang Wang. An efficient materialized views selection algorithm based on pso. In *ISA 2009*, pages 1–4. IEEE, 2009.

[10] Qingzhou Zhang, Xia Sun, and Ziqiang Wang. An efficient ma-based materialized views selection algorithm. In *CASE 2009*, pages 315–318. IEEE, 2009.

[11] Jiratta Phuboon-ob and Raweewan Auepanwiriyakul. Two-phase optimization for selecting materialized views in a data warehouse. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 1(1):119–123, 2007. ISSN eISSN:1307-6892. URL http://waset.org/Publications?p=1.

[12] Roozbeh Derakhshan, Bela Stantic, Othmar Korn, and Frank Dehne. Parallel simulated annealing for materialized view selection in data warehousing environments. In *ICA3PP 2008*, pages 121–132. Springer, 2008.

[13] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. Clustering-based materialized view selection in data warehouses. In Yannis Manolopoulos, Jaroslav Pokorný, and Timos K. Sellis, editors, *Advances in Databases and Information Systems*, pages 81–95, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-37900-3.

[14] Kamel Aouiche and Jérôme Darmont. Data mining-based materialized view and index selection in data warehouses. *Journal of Intelligent Information Systems*, 33(1):65–93, 2009.

[15] T. V. Vijay Kumar, Archana Singh, and Gaurav Dubey. Mining queries for constructing materialized views in a data warehouse. In *Advances in Computer Science, Engineering & Applications*, pages 149–159, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-30111-7.

[16] Rajib Goswami, DK Bhattacharyya, and Malayananda Dutta. Materialized view selection using evolutionary algorithm for speeding up big data query processing. *Journal of Intelligent Information Systems*, 49 (3):407–433, 2017.

[17] Jesus Camacho Rodriguez. Materialized views in apache hive 3.0. https://cwiki.apache.org/confluence/display/Hive/Materialized+views, 2018. Last accessed: 2018.10.15.

[18] Deepak Vohra. Apache hive. In *Practical Hadoop Ecosystem*, pages 209–231. Springer, 2016.

[19] Rada Chirkova, Alon Y Halevy, and Dan Suciu. A formal perspective on the view selection problem. In *VLDB 2001*, volume 1, pages 59–68, 2001.

[20] Patrick E O'Neil, Elizabeth J O'Neil, and Xuedong Chen. The star schema benchmark (revision 3, june 5, 2009). https://www.cs.umb.edu/~poneil/StarSchemaB.PDF, 2009.

[21] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Clustering analysis. In *The elements of statistical learning: Data mining, inference and prediction*, chapter 14, pages 501–520. Springer series in statistics, New York, 2009.

[22] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *Computing Research Repository (CoRR)*, abs/1109.2378, 2011. URL http://arxiv.org/abs/1109.2378.

[23] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20(1):53–65, 1987.

[24] Deepak Vohra. Apache parquet. In *Practical Hadoop Ecosystem*, pages 325–335. Springer, 2016.

[25] Deepak Vohra. Apache avro. In *Practical Hadoop Ecosystem*, pages 303–323. Springer, 2016.

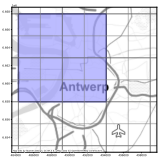
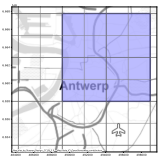
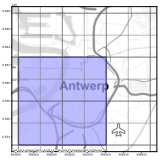
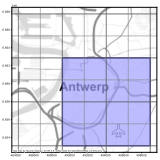
[26] Leandro Ordonez-Ante, Thomas Vanhove, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert, and Filip De Turck. Dynamic data transformation for low latency querying in big data systems. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2480–2489, Dec 2017. doi: 10.1109/BigData.2017.8258206.

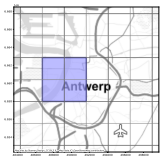
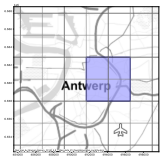
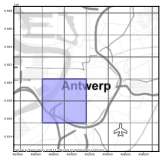
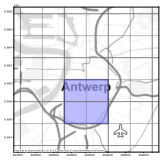
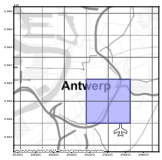
B

Query set used for the benchmark evaluation of the EXPLORA-LD platform

⋆⋆⋆

Table B.1: Visual queries used for the performance evaluation

Query polygon	Bounding box coordinates	Slippy tiles covered (zoom/x/y)
	*Left*: 4° 21’ 2.1096’’ E	13/4196/2734
	Bottom: 51° 12’ 24.7788’’ N	13/4195/2733
	Right: 4° 26’ 18.5172’’ E	13/4195/2734
	Top: 51° 15’ 42.894’’ N	13/4196/2733
	*Left*: 4° 23’ 40.3116’’ E	13/4196/2734
	Bottom: 51° 12’ 24.7788’’ N	13/4196/2733
	Right: 4° 28’ 56.7192’’ E	13/4197/2733
	Top: 51° 15’ 42.894’’ N	13/4197/2734
	*Left*: 4° 21’ 2.1096’’ E	13/4196/2735
	Bottom: 51° 10’ 45.6348’’ N	13/4195/2734
	Right: 4° 26’ 18.5172’’ E	13/4195/2735
	Top: 51° 14’ 3.8652’’ N	13/4196/2734
	*Left*: 4° 23’ 40.3116’’ E	13/4197/2735
	Bottom: 51° 10’ 45.6348’’ N	13/4196/2734
	Right: 4° 28’ 56.7192’’ E	13/4196/2735
	Top: 51° 14’ 3.8652’’ N	13/4197/2734
	*Left*: 4° 22’ 21.2124’’ E	14/8392/5468
	Bottom: 51° 13’ 14.3292’’ N	14/8391/5467
	Right: 4° 24’ 59.4144’’ E	14/8391/5468
	Top: 51° 14’ 53.3868’’ N	14/8392/5467
	*Left*: 4° 23’ 40.3116’’ E	14/8393/5468
	Bottom: 51° 13’ 14.3292’’ N	14/8392/5467
	Right: 4° 26’ 18.5172’’ E	14/8392/5468
	Top: 51° 14’ 53.3868’’ N	14/8393/5467

	*Left*: 4° 24’ 59.4144’’ E	14/8393/5468
	Bottom: 51° 13’ 14.3292’’ N	14/8393/5467
	Right: 4° 27’ 37.6164’’ E	14/8394/5467
	Top: 51° 14’ 53.3868’’ N	14/8394/5468
	*Left*: 4° 22’ 21.2124’’ E	14/8392/5469
	Bottom: 51° 12’ 24.7788’’ N	14/8391/5468
	Right: 4° 24’ 59.4144’’ E	14/8391/5469
	Top: 51° 14’ 3.8652’’ N	14/8392/5468
	*Left*: 4° 24’ 59.4144’’ E	14/8394/5469
	Bottom: 51° 12’ 24.7788’’ N	14/8393/5468
	Right: 4° 27’ 37.6164’’ E	14/8393/5469
	Top: 51° 15’ 42.894’’ N	14/8394/5468
	*Left*: 4° 22’ 21.2124’’ E	14/8392/5470
	Bottom: 51° 11’ 35.214’’ N	14/8391/5469
	Right: 4° 24’ 59.4144’’ E	14/8391/5470
	Top: 51° 13’ 14.3292’’ N	14/8392/5469
	*Left*: 4° 23’ 40.3116’’ E	14/8392/5469
	Bottom: 51° 11’ 35.214’’ N	14/8392/5470
	Right: 4° 26’ 18.5172’’ E	14/8393/5469
	Top: 51° 13’ 14.3292’’ N	14/8393/5470
	*Left*: 4° 24’ 59.4144’’ E	14/8394/5469
	Bottom: 51° 11’ 35.214’’ N	14/8393/5469
	Right: 4° 27’ 37.6164’’ E	14/8393/5470
	Top: 51° 13’ 14.3292’’ N	14/8394/5470