Hindawi Publishing Corporation
International Journal of Digital Multimedia Broadcasting
Volume 2012, Article ID 716780, 12 pages
doi:10.1155/2012/716780

Research Article

Multi-Objective Genetic Algorithm for Task Assignment on Heterogeneous Nodes

Carolina Blanch Perez del Notario, Rogier Baert, and Maja D'Hondt

SSET Department of IMEC, Kapeldreef 75, 3001 Leuven, Belgium

Correspondence should be addressed to Carolina Blanch Perez del Notario, blanch@imec.be

Received 30 April 2011; Revised 5 September 2011; Accepted 20 September 2011

Academic Editor: Yifeng He

Task assignment in grid computing, where both processing and bandwidth constraints at multiple heterogeneous devices need to be considered, is a challenging problem. Moreover, targeting the optimization of multiple objectives makes it even more challenging. This paper presents a task assignment strategy based on genetic algorithms in which multiple and conflicting objectives are simultaneously optimized. Specifically, we maximize task execution quality while minimizing energy and bandwidth consumption. Moreover, in our video processing scenario; we consider transcoding to lower spatial/temporal resolutions to tradeoff between video quality; processing, and bandwidth demands. The task execution quality is then determined by the number of successfully processed streams and the spatial-temporal resolution at which they are processed. The results show that the proposed algorithm offers a range of Pareto optimal solutions that outperforms all other reference strategies.

1. Introduction

Nowadays multimedia applications such as multicamera surveillance or multipoint videoconferencing, are increasingly demanding both in processing power, and bandwidth requirements. In addition, there is a tendency towards thin client applications where the processing capacities of the client device are reduced and the tasks are migrated to more powerful devices in the network.

In this respect, grid computing can integrate and make use of these heterogeneous computing resources which are connected through networks, overcoming the limited processing capabilities at a client's device.

In the context of distributed media processing we can think of scenarios such as video control rooms where multiple video streams are processed and simultaneously displayed. One way to downscale the processing and bandwidth requirements at the displaying device is by transcoding the video streams at the servers to lower temporal or spatial resolutions. This is done, however, at the cost of a degraded perceived video quality and an increased processing cost at the server. Therefore, in grid computing we may need to optimize and trade off multiple objectives, targeting for instance quality maximization of the stream execution and minimization of the energy consumption on the client/servers simultaneously. In this respect, implementing a suitable strategy for task assignment/scheduling becomes crucial for achieving a good performance in grid computing. This subject has been thoroughly studied in literature, and various heuristic approaches have been widely used for scheduling. In particular, Genetic Algorithms (GAs) have received much attention as robust stochastic search algorithms for various optimization problems. In this context, works such as [1–3] have used Genetic Algorithms to approach task scheduling within a device. In [1, 2], genetic algorithms are combined with heuristics such as "Earliest Deadline First" to assign tasks onto multiple processors. The work in [3] extends the analysis to heterogeneous processors where the algorithm can also determine the optimal number of processors, given a number of tasks. In [4, 5], the work is extended to multiple processing nodes in the network. This way, in [4], a thorough study is done on the performance of different GA operators on a scheduling problem on grid computing systems. Heterogeneous processors are considered with the target to minimize the makespan and flow time of the tasks.

The authors in [5] use also GA to address a similar objective in grid computing. As in [4] neither data transmission nor resource cost is considered.

In [6], a combination between GA and ACO algorithms is presented for task scheduling among multiple nodes but no bandwidth considerations are made. In [7], the authors use evolutionary fuzzy systems to solve job scheduling in decentralized computational grids where jobs between grid clusters are exchanged. The average response time per job is minimized but again the overhead of data transfers is not considered. The work in [8] also uses a genetic-based algorithm to assign multiple tasks in grid computing and minimize the make-span in the task execution but unlike previous works, the transmission time to the processing node is considered. Similar considerations are taken in [9] where the authors propose a strategy based on the Ant Colony Optimization algorithm (ACO) to effectively assign tasks to computing resources, given the current load conditions of each computing resource and network status.

Note that the presented works only consider single objective optimization. It is in works such as [10–16] where GAs are used to address multiple objectives in the resolution of scheduling problems. This way, in [10], the authors address the Job Shop scheduling problem while targeting multiple objectives, namely, minimal make-span and minimal machine workload. However, the case study addressed is very simple and only homogeneous processors are considered. The work in [11] addresses the flow-shop scheduling problem with an adaptive genetic algorithm. Pareto fronts are used to guide a multiple-objective search: the total completion time and total tardiness. As in our work, multiple objectives are addressed, however, task assignments at system level and bandwidth limitations are not considered. The authors in [12] consider heterogeneous processors and a multiobjective GA targets in this case the minimization of the make-span, flow-time, and reliability. The work in [13] is an early work on applying GA for multiobjective optimization, in this case to minimize both task execution and power during cosynthesis of distributed embedded systems. In [14], the performance of a NSGA-II-based evolutionary algorithm is compared to the true Pareto front of solutions found with the Branch and Bound method. The objectives targeted are the make-span and the task completion time. Finally, in [15] the method of particle swarm optimization is combined with evolutionary method outperforming typical genetic algorithms.

However, in none of these works [12–15], there is any consideration made over bandwidth constraints. Only in [16] the network resources in terms of bandwidth and latency are considered. In this case, an Ant Colony Optimization algorithm is developed to address grid scheduling.

In our approach, we use GAs to target multiple objectives for task assignment in grid computing and we consider bandwidth availability between nodes. Moreover, in comparison with all related work presented, in our analysis, we introduce an extra dimension on the task assignment problem by considering the downscaling of the video streams to lower spatial/temporal resolution. This offers a tradeoff between bandwidth and processing constraints on one hand and perceived video quality on the other hand. By doing this the effective system capacity to process tasks is increased while a graceful degradation of the video stream quality is allowed. Additionally, we target multiple objectives such as task quality maximization, client's energy minimization, and minimization of the bandwidth usage.

The rest of the paper is structured as follows. Section 2 describes the scenarios for distributed media processing considered. Section 3 describes the basic strategies for task assignment as well as the strategy for quality maximization while Section 4 introduces the strategy based on genetic algorithms. We present the results and compare the performance of the different strategies in Section 5. Finally, Section 6 concludes this work.

2. Scenario of Distributed Media Processing

In the context of distributed video processing, we are considering a scenario such as the one of a video control room. Several video contents are streamed towards the client device, where the content is visualized, while the required video processing can be distributed between the client and other processing nodes such as servers.

We assume all processing nodes to be heterogeneous with a different amount of processing resources, such as CPUs and GPUs. In addition, we assume that the client's device has more limited processing capacities than the server nodes. Concretely, we consider 4CPUs and 1GPU at each server node, while only 2CPUs at the client node. We assume moreover that multiple codec implementations for these different processor types are available. To overcome the limited processing at the client node, we perform distributed processing over other nodes in the network. In this case, the decoding task is executed at a server, and the resulting output (raw video) is transmitted to the client's device. Note that this highly increases the bandwidth requirements, which should fit in the maximum available bandwidth towards the client that we assume of 1 Gbps and shared from any server node to the client node. Therefore, to fit both processing and bandwidth requirements, one possibility is to trans-code (decode and reencode at a lower temporal or spatial resolution) the video streams at the server's side. This lowers both its bandwidth and decoding processing requirements at the end device at the cost of a reduced perceived quality and increased server processing.

The following section describes the task assignment strategies used in the scenario described.

3. Single Objective Assignment Strategies

An efficient task assignment strategy is a key element in the context of distributed grid computing. In this section, we describe the assignment strategies that we implement for comparison with our evolutionary-based approach.

Round-Robin Strategy (RR). The stream processing tasks are assigned in turns on the different available processing elements, that is, client device and server nodes.

Max-Min Heuristic (MM). This is a well-known heuristic [17] that assigns the most demanding stream processing tasks first on the processor that is going to finish them the earliest.

TransCode-All Strategy (TA). We assign all video streams to be spatially trans-coded at the server nodes. This lowers the processing requirements for decoding at the client devices while it also reduces the bandwidth usage. However, this happens at the cost of a reduced quality of the video streams. As trans-coding is an intensive task (decoding plus encoding) the trans-coding of the streams is evenly distributed among the available servers (by means of round robin) to avoid processing overload of a server.

Strategy Maximum Quality (MaxQ). In addition to the presented strategies, we implement a strategy that targets the maximization of the quality of the stream assignment. We describe this strategy next for the case of 1 server and 1 client node, where P_{client} and P_{server} are the total processing cost at client and server for the current task assignment, and $P_{\text{client}}^{\max}$ and $P_{\text{server}}^{\max}$ are the maximum processing capacity at the client node and server respectively. In a similar way, BW is the bandwidth required by the current task assignment while $\text{BW}^{\max}$ corresponds to the maximum available bandwidth.

We consider that the assignment and execution, t_i, of video stream "i" can take one of the following values:

- c: stream assigned to be decoded at the client at original temporal and spatial resolution.
- s: stream assigned to be decoded at the server at original temporal and spatial resolution.
- temp: stream transcoded to lower temporal resolution at the server.
- spat: stream transcoded to lower spatial resolution at the server.
- comb: stream transcoded to both lower temporal and spatial resolution at the server.

This way, our task assignment consists of a set of $t_i \in \{c, s, \text{temp}, \text{spat}, \text{comb}\}$ for every $i \in \{1, \ldots, N\}$ where N is the total number of video streams to be processed.

We want to find a task assignment solution whose bandwidth and processing demands at client and server fit within the bandwidth and processing constraints:

$$\begin{aligned} \text{BW} &= \sum_i \text{BW}_i \leq \text{BW}^{\max}, \\ P_{\text{client}} &= \sum_i P_{i,\text{client}} \leq P_{\text{client}}^{\max}, \\ P_{\text{server}} &= \sum_i P_{i,\text{server}} \leq P_{\text{server}}^{\max}, \\ \forall i \mid t_i &\in \{c, s, \text{temp}, \text{spat}, \text{comb}\}. \end{aligned} \tag{1}$$

Algorithm 1 is described in the following paragraphs and table.

In Step 1, the algorithm assigns as many stream decoding tasks as possible to execute on the client device; this number of tasks is constrained by the processing power at the device.

Step 1: *Maximize* $\{k \mid t_i = c\ \nabla i = \{1, \ldots, k\}, P_{\text{client}} \leq P_{\text{client}}^{\max}\}$
Step 2: *Set* $t_i = s\ \nabla i = \{(k+1) \cdots N\}$
WHILE ($P_{\text{client}} \geq P_{\text{client}}^{\max}$ or $P_{\text{server}} \geq P_{\text{server}}^{\max}$ or $\text{BW} \geq \text{BW}^{\max}$)
  Step 3: IF $\exists i = \max_i\{\text{BW}_i \mid t_i = s\}$
    THEN $t_i = temp$
  ELSE IF $\exists i = \max_i\{\text{BW}_i \mid t_i = temp\}$
    THEN $t_i = spat$
  ELSE IF $\exists i = \max_i\{\text{BW}_i \mid t_i = spat\}$
    THEN $t_i = comb$
  END
  Repeat Step 3 *until* $\text{BW} \leq \text{BW}^{\max}$
  Step 4: IF ($P_{\text{server}} \geq P_{\text{server}}^{\max}$ *and* $t_i = comb\ \nabla i = \{1, \ldots, N\}$
    THEN ***STOP***
  ELSEIF $P_{\text{client}} \geq P_{\text{client}}^{\max}$ *THEN*
    IF $\exists i \mid t_i = c$ *THEN*
      Set $t_i = s$
    ELSE ***STOP***
  END
END

Algorithm 1

In Step 2, the remaining tasks, exceeding the processing power at the client device, are assigned for processing at the server.

Then, we check if the current assignment meets bandwidth and processing constraints. While either bandwidth or processing constraints at client or server are not met, the algorithm will gradually transcode video tasks to lower temporal or spatial resolution at the server (done in Step 3) or will migrate some of the decoding tasks from the client device to the server (done in Step 4). This process continues till the assignment fits the system bandwidth and processing constraints.

In Step 3 we proceed as follows.

(i) Find those stream which are currently assigned at original temporal and spatial resolution to the server ($t_i = s$). From those, pick the stream with the biggest BW demand and transcode it to lower temporal resolution ($t_i = \text{temp}$).

(ii) If there are no streams available at full temporal resolution, then we take a stream at lowered temporal resolution ($t_i = \text{temp}$) with the highest bandwidth demand and transcode it to a lower spatial resolution ($t_i = \text{spat}$).

(iii) If all streams have been spatially transcoded, then we pick one of them (at highest bandwidth) and transcode it both temporally and spatially ($t_i = \text{comb}$).

Note that at this point (Step 3), we are trying to find those stream tasks (t_i) that demand the maximum bandwidth (generally also the highest processing) in order to reduce the bandwidth demands by trans-coding the minimum amount of streams.

This procedure is repeated till the bandwidth constraint is met.

Finally, in Step 4, if the processing constraints at the client are exceeded we migrate one client task to the server side. If at the server's side the processing constraints are not met and all streams have been spatially and temporally transcoded, the assignment loop is stopped. It is not possible to downscale the stream tasks further, and therefore we cannot find an assignment that satisfies all constraints while processing all streams.

3.1. Evaluation of Stream Assignment Cost. If the task assignment exceeds any of the system constraints in (1), then the execution of some tasks will fail. This way, an individual stream processing task fails when it does not fit within the available processing or bandwidth resources. For instance, the node where the stream is assigned may not have sufficient processing resources, or even if the processing is completed at the server, the available bandwidth could be insufficient to deliver the server's output to the client causing the stream processing to fail.

Related to this, we can attach to each assignment solution a corresponding cost in terms of end video quality, bandwidth usage, and energy consumption. This cost is determined by how many stream tasks are successfully completed and how (on which device and at what spatial-temporal resolution) they are executed.

Therefore, for a specific stream assignment solution we first need to estimate which stream processing tasks can be successfully completed and which will fail due to not meeting current processing and bandwidth constraints. Then, depending on the specific execution of each individual stream processing, we can attach a cost, in terms of quality, consumed bandwidth, and energy at the client's side, as defined in Table 1.

Note that the data in Table 1 corresponds to streams of Standard Definition (SD, 720 × 480 pixels). We also consider streams of Common Intermediate Format (CIF, 352 × 288 pixels) and High Definition (HD, 1920 × 1080 pixels) where bandwidth and energy costs are scaled accordingly with respect to SD resolution.

A stream processing fails when it does not fit within the available processing, or bandwidth resources. For instance, the node where the stream is assigned may not have sufficient processing resources or even if the processing is completed at the server, the available bandwidth could be insufficient to deliver the server's output to the client causing the stream processing to fail.

We attach successfully processed streams a quality value of 1 when the content is displayed at the client at its original temporal and spatial resolution. If the video stream is downscaled to a lower temporal/spatial resolution in order to fit bandwidth or processing constraints, the perceived video quality will be slightly degraded, and therefore, we attach a lower quality value. This favors that to maximize the streams quality, assignment solutions where the original spatial and temporal resolution of the streams are kept are preferred. Note that the quality value of any stream at its original resolution (CIF, SD, or HD) is identical; only in transcoding, we consider the quality degraded; that is, an HD-streamed spatially transcoded (to SD) is attached a 0.8 quality (distortion of 0.2) while a stream at original SD resolution is attached the maximum quality of 1 (0 distortion).

Table 1: Definition of task execution costs (SD resolution).

Stream Execution	Quality TQ_i	Bandwidth TBW_i	Energy TE_i
Failed execution	0	0 Mbps	0
Decoding at client	1	20 Mbps	1
Decoding at server	1	146 Mbps	0
Temporal transcoding	0.9	10 Mbps	0.5
Spatial transcoding	0.8	7 Mbps	0.25
Temporal and Spatial transcoding	0.7	3.5 Mbps	0.12

The bandwidth cost per stream is also dependent on how the stream processing is performed. This way, if decoding is performed at the server's side, the stream is transmitted raw to the client, which highly increases the bandwidth requirements. On the contrary, if the stream is transcoded at a server to a lower spatial or temporal resolution, the bandwidth requirements are reduced. For the sake of simplicity, we assume the same bandwidth cost for all video streams with the same spatial-temporal resolution. In addition, we consider that reducing the temporal resolution from 30 frames per second to 15 approximately reduces the bandwidth by half. Similarly, we assume that reducing the spatial resolution to the immediate lower resolution roughly reduces the bandwidth to approximately one-third of the original resolution.

Finally, in terms of energy/processing cost at the client's device, we assume that the energy cost is negligible when the video decoding task is executed on a server, and the raw output video stream is merely transmitted to the client device for display. When the decoding task is executed at the client, the corresponding energy cost is dependent on the temporal and spatial resolution of the decoded stream. We assume that decoding a video sequence at 15 fps requires approximately half of the processing/energy than decoding the same sequence at 30 fps. In a similar way, when the spatial resolution is lowered, for example from SD to CIF, we can roughly assume 1/4 of the decoding energy costs. Finally, the combination of lowering temporal and spatial resolution corresponds to a decoding cost of 18 of the original resolution. In case of a failed task execution, we assume no processing effort at the client's side and therefore no energy consumption. Note that the values in Table 1 are taken as approximate and reference values and have no impact on the relative performance of the assignment strategies.

To obtain the total quality TQ, bandwidth TBW, or energy cost TE for the complete assignment, we simply need to sum all individual stream processing costs as follows:

$$\text{Quality TQ} = \sum_{i=1}^{N} \text{TQ}_i \tag{2}$$

or equivalently

$$\text{Distortion TDist} = \sum_{i=1}^{N} \text{Max Quality} - \sum_{i=1}^{N} \text{TQ}_i, \tag{3}$$

$$\text{Bandwidth TBW} = \sum_{i=1}^{N} \text{TBW}_i, \tag{4}$$

$$\text{Energy TE} = \sum_{i=1}^{N} \text{TE}_i, \tag{5}$$

where N is the total number of streams considered in the assignment.

4. Multiobjective Genetic Algorithm

The heuristic and strategies presented in the previous section target at most one single objective optimization. However in practice we may want to optimize multiple objectives simultaneously. For instance, we may need to maximize the video streams quality while minimizing the bandwidth usage and the energy cost at the client. This multiobjective optimization is challenging, especially when multiple heterogeneous nodes and multiple ways of processing the streams (decoding, trans-coding) are considered. To achieve this, we base ourselves on genetic algorithms and use the concept of Pareto fronts of solutions. This allows us to obtain a set of Pareto optimal assignment solutions from which we can choose the solution that best meets the constraints or our preferences towards a certain objective. In addition, a genetic algorithm is a flexible tool where the target objective can be easily modified. The remainder of this section describes how the genetic algorithm is implemented.

4.1. Genetic Algorithm Structure. A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution. In a genetic algorithm, a population of strings (called chromosomes), which encode candidate solutions (called individuals of the population) to an optimization problem, evolves toward better solutions. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Generally, each generation of solutions improves the quality of its individuals. The algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached. The structure of our genetic algorithm can be summarized as follows.

Step 1. Initialize the population of chromosomes.

Step 2. Evaluate each chromosome with the *fitness* function.

Step 3. Crossover operation: select parents according to fitness and create new children chromosomes.

Step 4. Random *mutation* of chromosomes.

Step 5. Elitist *selection:* retain fittest chromosomes among those of the old generation and the new ones resulting from Steps 3 and 4.

Step 6. Repeat Steps 2 to 5 till termination condition is reached.

4.2. Representation of the Solution Domain. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. In our case, we use a decimal representation. Each possible stream assignment solution is represented as a chromosome, which is composed of several gens. In our case, the length of the chromosome is equal to the number of streams that need to be scheduled in the system. Each of the genes in the chromosome represents the node that is going to process the stream and how it is going to be processed. Table 2 gives a description of the meaning of each possible gene value in the chromosome. In this example, the available processing nodes are the client device S0 and two server nodes S1 and S2.

Figure 1 shows an example of chromosome (assignment solution). By looking at the gene description in Table 2, we can see that the corresponding stream task assignment would be as follows: the first two streams ("2") are decoded at server S1, the third and fourth streams ("1") are executed at the client device S0, the next three streams are transcoded at server S2 to a lower spatial resolution ("8"), and the last stream is processed at server S1 where it is transcoded to lower temporal resolution ("3").

We now detail how to compute the cost of the assignment solution in Figure 1. Assuming all streams are of SD resolution, the corresponding cost of this assignment according to Table 1 would be the following.

For streams 1 and 2 decoded at the server at full resolution and transmitted raw to the client:

$$\begin{aligned} \text{BW}_1 = \text{BW}_2 &= 146\,\text{Mbps}, \\ \text{TQ}_1 = \text{TQ}_2 &= 1, \\ \text{TE}_1 = \text{TE}_2 &= 0. \end{aligned} \tag{6}$$

For streams 3 and 4 decoded at the client device at full resolution:

$$\begin{aligned} \text{BW}_3 = \text{BW}_4 &= 20\,\text{Mbps}, \\ \text{TQ}_3 = \text{TQ}_4 &= 1, \\ \text{TE}_3 = \text{TE}_4 &= 1. \end{aligned} \tag{7}$$

For streams 5, 6, and 7 spatially transcoded at a server:

$$\begin{aligned} \text{BW}_5 = \text{BW}_6 = \text{BW}_7 &= 7\,\text{Mbps}, \\ \text{TQ}_5 = \text{TQ}_6 = \text{TQ}_7 &= 0.8, \\ \text{TE}_5 = \text{TE}_6 = \text{TE}_7 &= 0.25 \end{aligned} \tag{8}$$

For stream 8 that is temporally transcoded at a server before being sent to the client:

$$\text{BW}_8 = 10\,\text{Mbps}, \qquad \text{TQ}_8 = 0.9, \qquad \text{TE}_8 = 0.5. \tag{9}$$

Therefore, the total value of the assignment in the three-dimension space is

$$\sum_i \text{BW}_i = 363\,\text{Mbps}, \qquad \sum_i \text{TQ}_i = 7.3, \qquad \sum_i \text{TE}_i = 3.25. \tag{10}$$

Table 2: Description of genes.

Gene value	Meaning on task execution
"1"	Decoded at client device S0
"2"	Decoded at S1 and transmitted to S0
"3"	Transcoded at S1 to lower temporal resolution
"4"	Transcoded at S1 to lower spatial resolution
"5"	Transcoded at S1 to lower spatial-temporal resolution
"6"	Decoded at S2 and transmitted to S0
"7"	Transcoded at S2 to lower temporal resolution
"8"	Transcoded at S2 to lower spatial resolution
"9"	Transcoded at S2 to lower spatial-temporal resolution

2	2	1	1	8	8	8	3

Figure 1: Example of chromosome.

4.3. Initialization of GA Population. In general terms we initialize the population of assignment solutions by random generation. We also include in the initial population solutions that contribute to distribute the tasks processing evenly among the existing processing elements as well as solutions that imply transcoding of all processing tasks (as this may facilitate convergence to suitable assignments in high load scenarios). By doing so, we are making sure that certain potentially useful features are present in the population.

With respect to the population size, its optimal value is highly dependent on the scenario dimensions. In our case, we experimented with populations of size 10 to 40 and determined experimentally that a population of size 30 was suitable for the considered scenarios.

In addition, in a dynamic scenario where the number of tasks to be processed may be varying over time, we can improve convergence by reusing previously found solutions as part of the initial population for a new scenario. For instance, if the stream tasks to be processed increase from N to $N+1$, then we can use the assignment solutions found for N streams as initial population for the assignment of $N+1$ streams, while a random assignment is added for the extra $N+1$th stream. This helps the algorithm converge and finds an optimal solution. Moreover, it helps preserve the previous assignment solution as it minimizes the change of tasks assignments in the system.

4.4. Fitness Function. The goal of the fitness function is to evaluate how good an assignment solution is with respect to the defined target objectives. If we consider the optimization of a single objective, for instance, maximization of the video quality, we can define the fitness function as the quality value of every assignment:

$$\text{Fitness} = \text{Quality} = \sum_{i=1}^{N} TQ_i \qquad (11)$$

N = total number of streams.

This way, the fittest assignments are those with the highest video stream quality.

If we are considering multiple objectives such as video quality, bandwidth usage, and energy consumption at the client device, a particular assignment solution will result in a certain value in these three axes. In other words, each assignment solution can be represented as a point in the multiobjective space with the objective values (TDist, TBW, and TE).

Each of these values is obtained as the sum of distortion, bandwidth and energy for all N stream tasks considered:

$$\text{TDist} = \sum_{i=1}^{N} \text{TDist}_i, \quad \text{TBW} = \sum_{i=1}^{N} \text{TBW}_i, \quad \text{TE} = \sum_{i=1}^{N} \text{TE}_i. \qquad (12)$$

With the task distortion related to the task quality as

$$\text{TDist}_i = \text{Max}\, Q_i - \text{TQ}_i. \qquad (13)$$

To minimize multiple objectives, we would like then to minimize the following function where $w1$, $w2$, and $w3$ are the weights for the different objectives:

$$f(\text{dist}, \text{BW}, E) = w1 * \text{TDist} + w2 * \text{TBW} + w3 * \text{TE}. \qquad (14)$$

Minimizing (14) is then equivalent to maximizing the following fitness function in our GA:

$$\text{Fitness} = \frac{1}{f(\text{dist}, \text{BW}, E)}. \qquad (15)$$

To avoid the need of weighting factors in the fitness function, we use the concept of Pareto points for the multiobjective optimization. Pareto points are optimal tradeoffs in the multiobjective space and obtaining the set of Pareto points from all assignment solutions is equivalent to exploring a range of weights in (14)-(15).

We are therefore interested in obtaining a range of Pareto optimal solutions in said multiobjective space. In addition, we evaluate the fitness of an assignment solution according to how close the solution is to a Pareto point or to the actual Pareto envelope.

In Figure 2, we can see that this favors that non-Pareto solutions like A, which lie close to the Pareto front, are selected to breed and mutate hopefully generating Pareto points within new areas of the Pareto front. This helps newer generations of Pareto front spread over the bidimensional space without losing diversity and concentrating around the specific area of the initial Pareto points.

To evaluate the fitness of each solution point, we compute the Euclidean distance from each point to the closest point in the hypothetical Pareto front. In a three-dimensional objective space, this is expressed as

$$\text{Fitness}(i) = \frac{1}{\min}(\text{Euclid}(i,j)) = \frac{1}{\min}\left(\sqrt{(i_1 - j_1)^2 + (i_2 - j_2)^2 + (i_3 - j_3)^2}\right) \quad \nabla j \in \text{Pareto front}, \qquad (16)$$

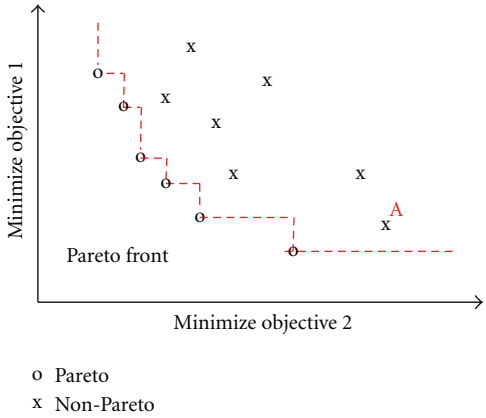

Figure 2: Pareto front of solutions.

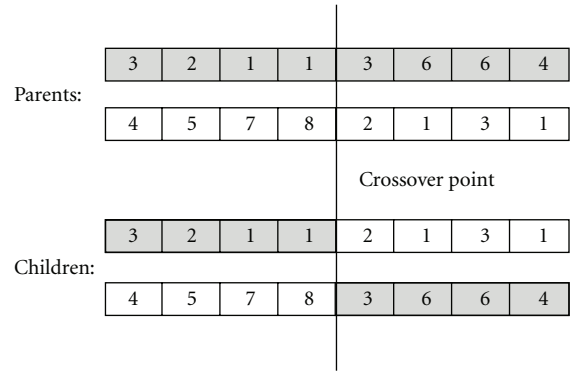

Figure 3: Single-point crossover.

where every point has the following representation in the three-dimensional multiobjective space $i = \langle i_1, i_2, i_3 \rangle$. The closer a point is to this Pareto front, the fitter it is considered. Note that Pareto points, already belonging to the Pareto envelope, have the minimum distance to it and therefore are assigned the highest fitness values. This guarantees that they are always kept for the next generation of the GA.

4.5. Selection of Individuals According to Fitness. During each successive generation, a proportion of the existing population of solutions is selected to breed a new generation. Individual solutions can be selected through a fitness-based process, where fitter solutions (as measured by the fitness function) are typically more likely to be selected. In our case, we do not use a probabilistic selection but an elitist selection, that is, the fittest members of the population are used to breed a new population. Moreover, after crossover and generation of new child solutions, we apply again elitist selection and retain in the solution space those solutions that are the fittest among the parents and the newly generated children solutions. We use the fitness function as described in the earlier section. In practice, this means that for the selection of the parent chromosomes during the crossover and mutation steps, we select the Pareto optimal points from the pool of chromosome/solutions (as these are the fittest points in our space) and from the non-Pareto points, we take the fittest ones.

The elitism we apply in the selection is similar to the one applied in the Nondominated sorting GA or NSGA [18] in the sense that the Pareto or nondominated solution points are identified and given the highest fitness value. Our approach differs however in the treatment of the non-Pareto solutions, in our case, the fitness of these points is computed based on the distance to the Pareto front, more similar to the Strength Pareto Evolutionary Algorithm (SPEA). With respect to SPEA [19], we apply a similar elitist selection, after both crossover and mutation as the Pareto solutions are always kept for the next generation. However, unlike SPEA, we do not maintain a fixed size for the Pareto population, and to avoid a too early convergence of the algorithm, we allow a high degree of mutation among the solutions in the Pareto front.

4.6. Evolution: Crossover and Mutation. The *crossover* step consists in creating a second generation of solutions from an initial population of solutions. For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the current population. By producing a "child" solution using the crossover, a new solution is created which typically shares many of the characteristics of its "parents". In our algorithm, crossover is implemented as single-point crossover. This is, a random point is selected in the parent chromosome, and the information before and after that point is exchanged and recombined between the two parents creating in this way two child solutions that share the parent's characteristics. The single-point crossover is represented in Figure 3.

Crossover is applied on a percentage of the population given by the *crossover rate*. Out of the new population of "parents" and "child" chromosomes, we apply an elitist selection and keep a population with size of the initial population with the fittest individuals. Generally, the average fitness of the new population will have increased, since only the best individuals from the first generation are selected for breeding. After crossover, the *mutation* step is implemented. The purpose of mutation in GAs is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. Mutation is implemented by randomly changing some of the genes in the chromosome with a probability given by the *mutation rate*.

We use a high mutation rate in our approach, as this helps the algorithm avoid local minima. Nevertheless, we do not risk losing good features of the solution space, thanks to the elitist selection applied after crossover or mutation, that is, the fittest solutions are always kept; therefore, if the mutated solutions are less fit than the original solutions, the original solutions are retained.

4.7. Parameter Selection for Genetic Algorithm. The way the Genetic Algorithm evolves towards fitter solutions is highly dependent on the parameter selection. In this respect, the percentage of the population on which the crossover and

Table 3: Assignment quality versus population size.

Nos. of tasks/population	10	20	30	40
10	10	19.7	28	35.9
20	10	19.9	28.4	36.8
30	10	19.9	28.7	36.5
40	10	19.9	28.4	36.6

Table 4: GA parameters.

Crossover rate	0.7
Mutation rate	0.3
Population size	30
Max generations	30

mutation steps are implemented is given by the crossover and mutation rate parameters, respectively. As explained in the previous section, a high mutation rate is selected to prevent the algorithm from a too early convergence and falling in local minima. A similar approach is taken in [14, 20]. Moreover, in [14], a similar crossover rate is also selected. Unlike [14] however, we require a lower population size and lower number of generations. This is possibly due to the fact that the elitist selection we implement helps speed up the convergence.

Table 3 shows the impact of the population size on the quality of the assignment solution found (with task quality as single objective) for a different number of tasks considered. We can first see that when the number of tasks increases it becomes more difficult to find an optimal assignment with high quality; this is due to the limited bandwidth and processing constraints in the system. This way, for 10 the tasks considered the assignment found guarantees perfect quality (equal to the number of tasks) while as the task increases the maximum quality attained differs from the perfect quality value.

In addition, we observed that to reach a high-quality solution, it is advisable to use a population with at least the same size than the number of tasks considered. Therefore, for the number of tasks considered in our scenarios, a population size of 30 individuals proves to be suitable. In this respect, we observed that bigger populations cause slow convergence and increased execution cost while small ones tend to evolve to less fit solutions.

In terms of number of iterations, we let the algorithm evolve during 30 generations. This value is experimentally found to be a good tradeoff in terms of achieving a good convergence while still having a reduced execution time.

Table 4 summarizes the parameter selection for the evolution of the Genetic Algorithm. The parameter values selected are also close to the suggested ones in [21].

4.8. Convergence of Genetic Algorithm. One way to analyze the convergence of our multiobjective GA is by measuring the area/volume under the Pareto front of solutions in the multiobjective space. The reason is that the minimization of several objectives in our GA translates to Pareto fronts becoming closer to all objective axes, in other words, Pareto fronts with lower areas/volumes underneath.

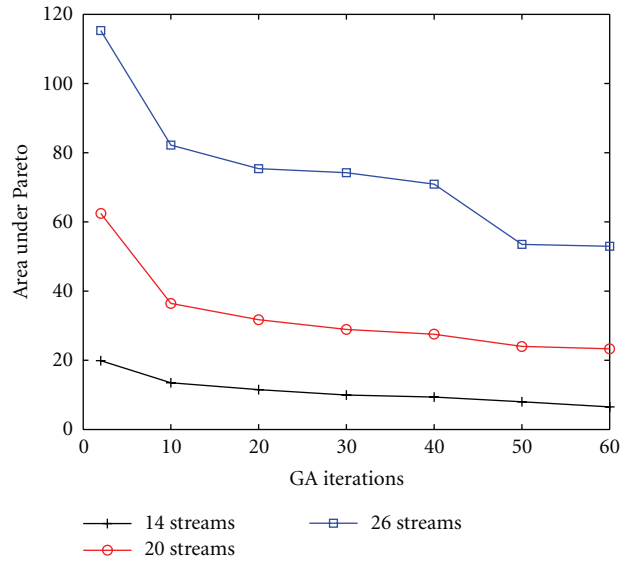

Figure 4: Convergence versus iterations.

Figure 4 shows the evolution of the area under the Pareto front versus the number of iterations for a different number of tasks in the system. In this case, the GA evolves with the objective to minimize both distortion and energy at the client. As we can see with an increasing number of iterations, the area below the Pareto front decreases. This indicates that the Pareto fronts obtained gradually improve and achieve lower values of distortion and energy.

As explained earlier, tenths of iterations are sufficient in our scenario to find good assignment solutions that outperform the reference methods.

4.9. Execution Cost of Genetic Algorithm. Our genetic algorithm is an in-house-developed Matlab code and does not form part of the Matlab Optimization toolbox. The code has not been optimized for speed and its average execution time for 10 iterations of the algorithm is in the order of a couple of seconds. In this respect, the computational cost of the reference methods such as MaxQ and Min-Max is almost negligible with respect to GA. However, these methods achieve suboptimal results and are not able to tackle a multiobjective optimization.

Note that genetic algorithms are subject to parallelization, which can speed up its execution considerably. Therefore, a more dedicated and optimized implementation of the algorithm exploiting parallelism would highly reduce its execution time. However, developing such algorithm is out of the scope of this paper. In previous work such as [14], we can observe similar execution times for the GA algorithm and a similar amount of tasks considered. In [22], an optimized implementation of a GA on a SUN 4/490 only requires 1 to 2 s to perform 1000 iterations showing that a dedicated implementation can reduce the execution time considerably. In this respect in our approach, the number of iterations needed

Table 5: Relative execution cost.

Nos. of tasks/population	10	20	30	40
10	8	9	8.5	8
20	14	15	18	19
30	20	20	30	32
40	27	29	40	41

is in the order of tenths of iterations, which would further reduce the execution time.

In addition, the computational load of the GA is marginal when compared to the high-computational load of any transcoding, decoding operation that takes place in the servers in our scenario. Therefore, the execution of the GA can be placed on such a server with high processing power elements such as a GPU.

Last but not least, in our cloud computing scenario we could expect that new stream processing tasks enter or leave the system not faster than every couple of minutes. Therefore, global or partial recomputations of the stream assignments are not frequently needed.

To give an indication of how the complexity of our algorithm scales, Table 5 shows the relative execution cost versus the number of tasks, and the size of GA population considered. We can see how, as expected, the execution cost increases almost proportionally with the size of the population. The increase with the number of tasks considered is much less noticeable. This has the advantage that scaling up the scenario to a higher number of tasks does not have a big impact on the GA algorithm complexity.

Note also that for large-scale problems, we could address the task scheduling problem in a hierarchical way, that is, tasks can be initially distributed locally among clusters of processing devices and within each cluster; GA can be applied to obtain the optimal assignment. This would limit the complexity increase of the GA optimization.

5. Experimental Results

In this section, we compare the performance of the different assignment strategies considered. We first focus on a single objective optimization, namely, quality, where we use the fitness definition in (11). Figure 5 shows the performance of each assignment strategy with respect to the system load (given by an increasing number of SD streams to be displayed at the client's device). We can see that from 4 video streams up to 10, both GA and MaxQ strategies outperform all others while achieving the maximum quality. In this respect, note that the maximum quality equals N, the total number of streams, as the maximum quality per stream is 1 (see Table 1). From 10 streams on, it is only the GA that achieves a close to optimal quality. This shows that the higher the load in the system is, the more advanced strategy we need to find a suitable assignment. Moreover, at high load transcoding with slight degradation of the stream quality becomes necessary in order to fit all streams into the available constraints.

We then focus on multiple objectives optimization for a specific system load. We consider the processing of 20 video

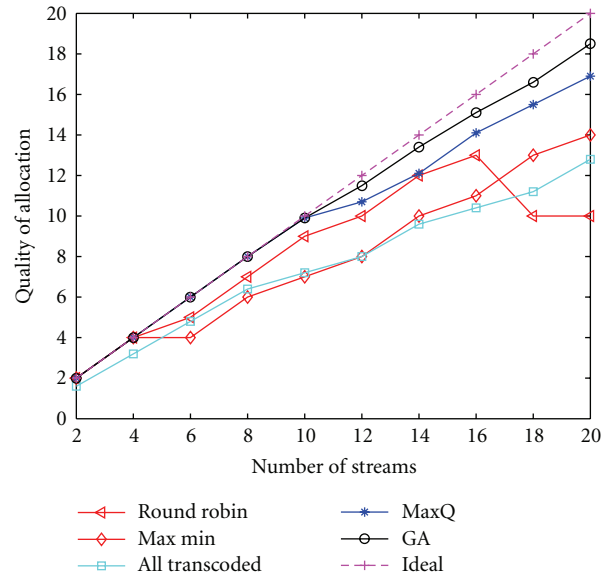

Figure 5: Performance of strategies versus system load.

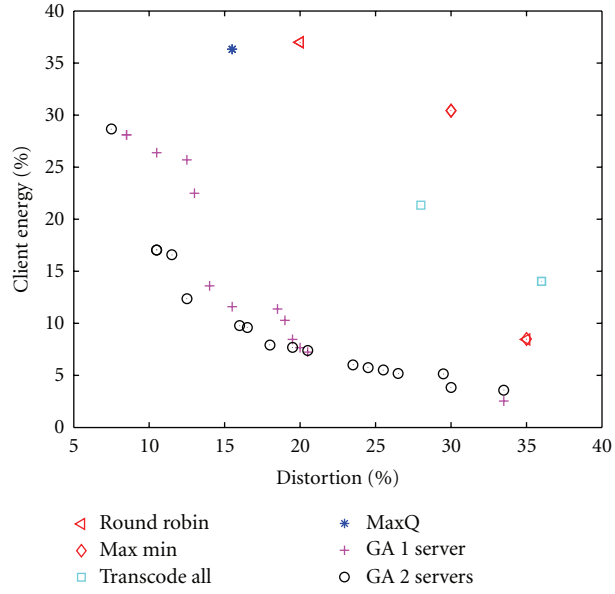

Figure 6: Energy-distortion tradeoffs.

streams of mixed spatial resolutions: CIF, SD, and HD at 30 fps. We compare the different strategies as well as the availability of one single server node with respect to two server nodes where tasks can be migrated to.

Figure 6 shows how the GA clearly outperforms all other strategies when targeting both quality maximization (equivalently distortion minimization) and minimization of energy at the client device. This way, the assignment solutions found by GA incur in lower energy at the client and lower distortion than the solutions found by the other strategies. We can see that in the GA Pareto front, some assignment solutions reach 7% of distortion while the MaxQ Strategy reaches around 15% distortion. Similarly, for the same

Table 6: Assignment strategies for HD/SD/CIF streams.

	Failed tasks	Decode @client	Decode @server	Temp trans	Spat trans	Temp & spat trans	Dist (%)	Energy (%)
RR	0/0/4	4/3/2	3/4/0	−/−/−	−/−/−	−/−/−	20	37
MM	0/0/6	7/6/0	0/1/0	−/−/−	−/−/−	−/−/−	30	30.4
TA	0/0/4	−/−/−	−/−/−	−/−/−	−/−/−	7/7/2	36	14
MaxQ	−/−/−	2/6/0	0/0/1	0/0/1	0/0/0	5/1/4	15.5	36.3
GA_1	−/−/−	4/0/1	2/5/0	1/2/0	0/0/1	0/0/4	8.5	28
GA_2	−/−/−	0/0/1	3/5/0	0/1/0	4/0/1	0/1/4	13	22.5
GA_3	−/−/1	0/1/0	5/5/0	1/0/0	1/0/0	0/1/5	15.5	11.6

distortion than MaxQ, the GA points lower the energy from 35% to roughly 10%.

Two sets of Pareto solutions are shown for the GA, one corresponding to the use of 1 server and another to the use of 2 servers. Naturally, having two servers available to process, the tasks allow the GA find better assignment solutions. Similarly, for the reference strategies, we show two points (assignment solution) for each strategy displayed. The two points correspond to the use of 1 server and 2 servers respectively, where generally the use of 2 servers achieves lower distortion but also higher energy consumption (as more streams can be processed).

Note that both distortion and energy values are given as percentages from the maximum possible distortion or energy. This way, the maximum energy cost at the client is defined as the cost of processing the decoding tasks of all streams at full spatial and temporal resolution, while the maximum distortion (100%) corresponds to a failed execution for all streams. In general, the relative distortion is defined as

$$\text{Distortion (\%)} = 100\% - \text{Quality (\%)}. \tag{17}$$

This way, from Table 1, successfully processing all N streams at its original spatial-temporal resolution would result in a maximum quality value of N (equivalent to 100% quality or 0% distortion). In a similar way, transcoding the temporal resolution of all N streams would correspond to a total quality of $0.9 * N$ (equivalent to a quality of 90% or distortion of 10%).

We can further analyze the assignment solutions found by the different strategies in Table 6. This table shows some of the assignment solutions of 1 server in Figure 6.

In this table, for the different kind of stream processing assignment, a distinction is made between the different original stream resolutions (HD/SD/CIF). This way, for example, in the assignment of the round robin (RR) strategy 4 CIF streams fail, while 4 HD streams, 3 SD, and 2 CIF are successfully decoded at the client, and 3 HD and 4 SD streams are decoded at the server. Finally, no streams are transcoded in this strategy. By analyzing Figure 6 and Table 6, we can see that strategies such as round-robin (RR) and max-min (MM) cannot find an assignment with a good tradeoff in terms of quality and energy cost. The reason is that none of these strategies considers either bandwidth constraints or transcoding. In this example, both round-robin and max-min succeed in distributing the load evenly among nodes and meet the processing constraints. However, decoding streams at the server nodes generates high bandwidth demands towards the client. This exceeds the bandwidth availability and causes failed processed streams (in particular the high bandwidth HD streams). Therefore in this case the use of 2 servers further degrades the performance as more streams are processed at the server's side, and this aggravates the bandwidth demand. This way, under high load some stream transcoding would be required to fit into both available processing and bandwidth constraints.

However, transcoding all streams at the server nodes (TA strategy) is neither an optimal assignment as we enforce the quality degradation of all streams. Moreover, transcoding tasks are processing intensive and exceed the processing capacity at the servers resulting in some failed stream processing. In this case, the use of 2 servers increases the overall processing capacity but the assignment remains quite suboptimal in terms of distortion and energy.

In contrast, the MaxQ strategy succeeds in finding an assignment with low distortion value (15%). However, its energy cost is relatively high (36%). In Table 6, we can see that all 20 streams are successfully processed, out of which 10 are transcoded temporally and spatially while 8 are decoded at the client device at full resolution. In this example, the processing capacities with one server are enough for the amount of transcoding involved, while the main limiting factor remains the bandwidth.

It is finally the GA that outperforms all strategies by addressing both objectives and finding a set of assignment solutions, that is, Pareto optimal in both senses. Moreover, having 2 server nodes available for processing allows the GA to find even better tradeoffs (lower Pareto curve) in terms of quality and energy. Note also that the set of GA solutions offers an energy range from 30% to 10% for low distortion values. This way, we can reduce the energy at the client by factor 3 by trading off some stream quality. This offers the flexibility to choose between different operating points at grid level according to how scarce the processing power and energy at the client is.

In Table 6, we only show a few of the set of Pareto optimal solutions found by the GA. We can see how GA_1 assignment solution reaches the lowest distortion (8.5%) at 20% lower energy cost than MaxQ strategy. This assignment distributes more evenly the streams between client and server and chooses to transcode temporally and spatially only 4 streams of HD resolution. The GA_2 assignment provides a tradeoff

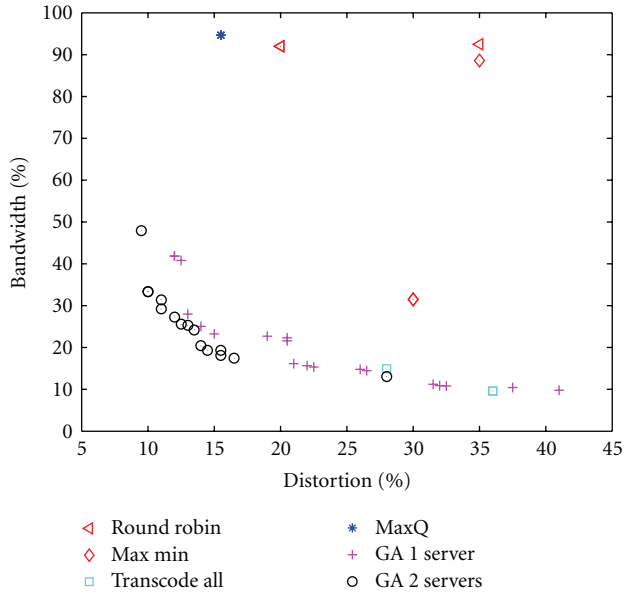

Figure 7: Bandwidth-distortion tradeoffs.

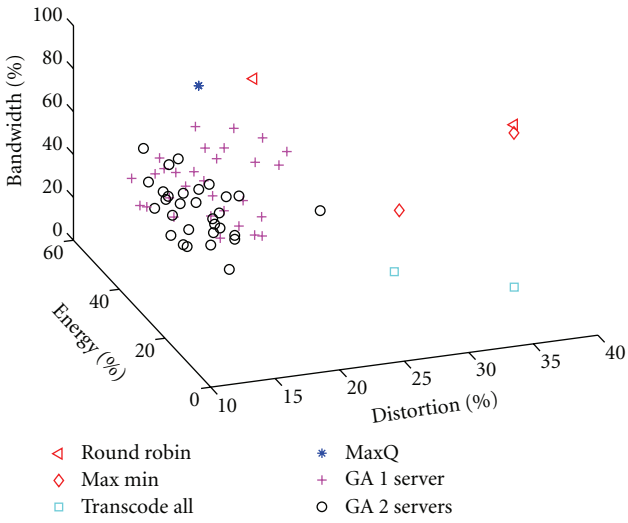

Figure 8: Pareto points in 3D space.

of slightly higher distortion (13%) but an energy cost 40% lower than that provided by MaxQ. Finally, GA_3 assignment provides similar distortion than MaxQ (15.5%) but it further reduces the energy consumption, demanding almost 70% less energy than MaxQ.

Figure 7 shows a similar comparison with respect to the bandwidth usage and distortion incurred by the different assignment solutions. In this case, the GA targets optimization of both distortion and bandwidth objectives and clearly outperforms all other strategies as shown by the lower distortion and bandwidth values obtained in the GA Pareto front. We can see in this respect how the MaxQ solution has a high bandwidth cost (close to 100%) while the GA solutions in the Pareto front require for the same distortion barely 20% of the maximum bandwidth. In addition, considering two

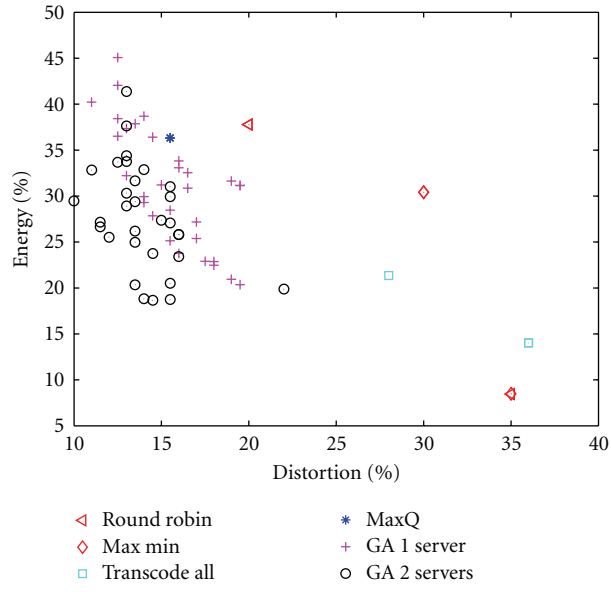

Figure 9: Projection onto energy-distortion space.

servers further allows a distortion and bandwidth reduction, possibly due to the fact that more processing power is available for transcoding the highly demanding HD streams.

Finally, Figure 8 shows the optimality of the different assignments in the three-dimensional space of distortion, bandwidth usage, and client's energy cost. The GA targets the minimization of these three objectives and the Pareto front becomes a Pareto surface.

As in the previous figures, the assignments found for 1 and 2 servers are displayed. Once again, we can see that the assignments found by the GA outperform all other strategies in terms of distortion, energy and bandwidth while at the same time, it provides a good tradeoff for all three objectives. Indeed, we can see that the GA solution points concentrate around lower distortion values (especially those corresponding to use of 2 servers), lower bandwidth, and lower energy values. For the sake of clarity, in Figure 9, we show the projection of Figure 8 on the two-dimensional space of energy and distortion where the better results of GA can be seen clearly.

In practice, both processing and bandwidth constraints may vary over time. Therefore, we may require a new stream assignment to fit the new constraints. One possible way to tackle this is by simply rerunning the assignment strategy. However, as the GA strategy already produces a set of assignment solutions with different energy-distortion-bandwidth tradeoffs, another possibility is to simply choose from the set of solutions a different operating point (assignment solution) that satisfies the new constraints. This way, if the current assignment solution requires a processing of 50% at the client's side, switching to a new assignment with 30% processing may be suitable for a more overloaded client device. We can also cope with variations in bandwidth or processing constraints by targeting more limiting constraints, for instance, 80% of the maximum bandwidth and maximum processing power. By doing so, the assignment is slightly overdimensioned and can cope with variations of up

to 20–25% above the current constraints. This would also help avoid too frequent task migrations in the system.

6. Conclusion

We have presented an evolutionary-based strategy for stream processing assignment in a client-cloud multimedia system where multiple heterogeneous devices are considered. In this context, we not only decide on which node each stream is assigned but we also consider the possibility of stream transcoding to a lower temporal or spatial resolution. This extends the system capacity at the cost of smooth quality degradation in the task execution.

Moreover, both processing capacities in the nodes and bandwidth availability are taken into consideration. The proposed strategy is highly flexible and can target multiple objectives simultaneously. It outperforms all other considered strategies while providing a wide range of tradeoffs in the assignment solutions.

Acknowledgment

The authors would like to thank Yiannis Iosifidis for his insights on genetic algorithms.

References

[1] Y.-W. Zhong and J.-G. Yang, "A hybrid genetic algorithm with Lamarckian individual learning for tasks scheduling," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC '04)*, pp. 3343–3348, October 2004.

[2] K. Dahal, A. Hossain, B. Varghese, A. Abraham, F. Xhafa, and A. Daradoumis, "Scheduling in multiprocessor system using genetic algorithms," in *Proceedings of the 7th IEEE Computer Information Systems and Industrial Management Applications (CISIM '08)*, pp. 281–286, June 2008.

[3] M. R. Miryani and M. Naghibzadeh, "Hard real-time multi-objective scheduling in heterogeneous systems using genetic algorithms," in *Proceedings of the 14th International CSI Computer Conference (CSICC '09)*, pp. 437–445, October 2009.

[4] J. Carretero, F. Xhafa, and A. Abraham, "Genetic algorithm based schedulers for grid computing systems," *International Journal of Innovative Computing, Information and Control*, vol. 3, no. 6, pp. 1053–1071, 2007.

[5] R. Entezari-Maleki and A. Movaghar, "A genetic algorithm to increase the throughput of the computational grids," *International Journal of Grid and Distributed Computing*, vol. 4, no. 2, 2011.

[6] J. Liu, L. Chen, Y. Dun, L. Liu, and G. Dong, "The research of ant colony and genetic algorithm in grid Task scheduling," in *Proceedings of the International Conference on MultiMedia and Information Technology, (MMIT '08)*, pp. 47–49, December 2008.

[7] A. Folling, C. Grimme, J. Lepping, and A. Papaspyrou, "Robust load delegation in service grid environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1304–1316, 2010.

[8] K.-M. Yu and C.-K. Chen, "An evolution-based dynamic scheduling algorithm in grid computing environment," in *Proceedings of the 8th International Conference on Intelligent Systems Design and Applications (ISDA '08)*, pp. 450–455, November 2008.

[9] A. Michalas and M. Louta, "Adaptive task scheduling in grid computing environments," in *Proceedings of the 4th International Workshop on Semantic Media Adaptation and Personalization (SMAP '09)*, pp. 115–120, December 2009.

[10] X. Yang, J. Zeng, and J. Liang, "Apply MGA to multi-objective flexible job shop scheduling problem," in *Proceedings of the International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII '09)*, pp. 436–439, December 2009.

[11] M. Basseur, F. Seynhaeve, and E.-G. Talbi, "Path relinking in Pareto multi-objective genetic algorithms," in *Proceedings of the 3rd International Conference on Evolutionary Multi-Criterion Optimization (EMO '05)*, pp. 120–134, March 2005.

[12] P. Chitra, S. Revathi, P. Venkatesh, and R. Rajaram, "Evolutionary algorithmic approaches for solving three objectives task scheduling problem on heterogeneous systems," in *Proceedings of the IEEE 2nd International Advance Computing Conference (IACC '10)*, pp. 38–43, February 2010.

[13] R. P. Dick and N. K. Jha, "MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 920–935, 1998.

[14] M. Camelo, Y. Donoso, and H. Castro, "MAGS—an approach using multi-objective evolutionary algorithms for grid task scheduling," *International Journal of Applied Mathematics and Informatics*, vol. 5, no. 2, 2011.

[15] F. S. Kazemi and R. Tavakkoli-Moghaddam, "Soving a multi-objective multi-mode resource-constrained project scheduling problem with particle swarm optimization," *International Journal of Academic Research*, vol. 3, no. 1, 2011.

[16] Y. Hu and B. Gong, "Multi-objective optimization approaches using a CE-ACO inspired strategy to improve grid jobs scheduling," in *Proceedings of the 4th ChinaGrid Annual Conference (ChinaGrid '09)*, pp. 53–58, August 2009.

[17] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," in *Proceedings of the 8th IEEE Heterogeneous Computing Workshop (HCW '99)*, pp. 30–44, San Juan, Puerto Rico, April 1999.

[18] N. Srinivas and K. Deb, "Multi-objective optimization using non-dominated sorting in genetic algorithms," *Evolution Computing*, vol. 2, no. 3, pp. 221–248, 1994.

[19] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[20] Y. Iosifidis, A. Mallik, S. Mamagkakis et al., "A framework for automatic parallelization, static and dynamic memory optimization in MPSoC platforms," in *Proceedings of the 47th Design Automation Conference, (DAC '10)*, pp. 549–554, June 2010.

[21] A. Y. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor systems: Comparative studies and performance issues," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 8, pp. 795–812, 1999.

[22] S. H. Hou, N. Ansari, and H. Ren, "Genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113–120, 1994.