SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2015; **45**:765–781
Published online 6 March 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2258

Implementation and performance evaluation of a scheduling algorithm for divisible load parallel applications in a cloud computing environment

Leila Ismail[1,*,†] and Latifur Khan[2]

[1]*College of IT, UAE University, Al-Ain, UAE*
[2]*Department of Computer Science, Erik Jonsson School of Engineering and Computer Science, University of Texas at Dallas Richardson, Richardson, Texas, USA*

SUMMARY

Cloud computing is an emerging technology in which information technology resources are virtualized to users in a set of computing resources on a pay-per-use basis. It is seen as an effective infrastructure for high performance applications. Divisible load applications occur in many scientific and engineering applications. However, dividing an application and deploying it in a cloud computing environment face challenges to obtain an optimal performance due to the overheads introduced by the cloud virtualization and the supporting cloud middleware. Therefore, we provide results of series of extensive experiments in scheduling divisible load application in a Cloud environment to decrease the overall application execution time considering the cloud networking and computing capacities presented to the application's user. We experiment with real applications within the Amazon cloud computing environment. Our extensive experiments analyze the reasons of the discrepancies between a theoretical model and the reality and propose adequate solutions. These discrepancies are due to three factors: the network behavior, the application behavior and the cloud computing virtualization. Our results show that applying the algorithm result in a maximum ratio of 1.41 of the measured normalized makespan versus the *ideal* makespan for application in which the communication to computation ratio is big. They show that the algorithm is effective for those applications in a heterogeneous setting reaching a ratio of 1.28 for large data sets. For application following the ensemble clustering model in which the computation to communication ratio is big and variable, we obtained a maximum ratio of 4.7 for large data set and a ratio of 2.11 for small data set. Applying the algorithm also results in an important speedup. These results are revealing for the type of applications we consider under experiments. The experiments also reveal the impact of the choice of the platforms provided by Amazon on the performance of the applications under study. Considering the emergence of cloud computing for high performance applications, the results in this paper can be widely adopted by cloud computing developers. Copyright © 2014 John Wiley & Sons, Ltd.

Received 15 September 2013; Revised 14 January 2014; Accepted 17 January 2014

KEY WORDS: distributed systems; cloud computing; divisible load application; autonomous computing; scheduling; perform

1. INTRODUCTION

Distributing the tasks of a parallel application to a set of distributing computing resources to decrease its overall execution time is very popular in distributed computing. Divisible load applications [1] or embarrassingly parallel applications are a class of distributed computing in which independent tasks are assigned to distributed resources with no synchronization and no inter-tasks communication [1–3]. Divisible load occurs in many scientific and engineering applications, such as search for a pattern, compression, join and graph coloring and generic search applications [4], multimedia and

*Correspondence to: Leila Ismail, College of IT, UAE University, 17551 Al-Ain, UAE.
†E-mail: leila@uaeu.ac.ae

video processing [5–7], image processing [8, 9] and data mining [10]. Distributed computation is meant to increase the efficiency of computing intensive applications, as computational load is distributed among a number of workers. However, this efficiency decreases due to the transfer cost of the parallel tasks from a master worker to the available computing workers and due to the computing latency involved by starting the remote execution.

Although many researchers have been seeking the best way to distribute chunks to increase the overall makespan of a divisible load application based on the network and computing resources capacities, and accounting for corresponding latencies in the distribution strategy, very few implement and evaluate the performance using real applications in a real cloud computing environment. In this work, we focus on the practice and experience of the implemented scheduling algorithm. We evaluate its performance, and we report on the learned lessons.

The cloud computing [11–13] is an emerging technology in which computing resources are virtualized to users on a pay-per-use basis. There is a great interest in porting parallel applications in cloud computing [14]. This is because in a cloud computing environment, virtual clusters with on-demand computing capacities are allocated to users to satisfy a Service Level Agreement [15]. In addition, those virtual clusters are elastic, which means they can dynamically scale up and down according to the applications' needs; that is, more cost-effective than using dedicated commodity clusters. However, running parallel application in a cloud computing environment face performance challenges, mainly due to the overhead introduced by the virtualization and the sharing of physical resources. In our previous work [16], we introduced a scheduling algorithm. An important limitation of the algorithm is that it considered fixed and equal latencies, which is not realistic in a cloud computing environment. In this work, we derive formulas to compute the chunk sizes of an application that should be distributed to a set of computing workers to obtain an optimal performance in a single round. The formulas take into consideration the network and computing capacities of the cloud and account for heterogeneous network and computing latencies. The single-round scheduling algorithm is chosen because it is the most widely used distribution algorithm by programmers thanks to its simplicity compared with a multiround algorithm [17, 18]. We implement and evaluate our algorithm with real applications using a well-known cloud computing environment, such as Amazon Elastic Cloud Computing (EC2) [19], and different instance types. Our aim is to reveal the performance of the selected applications by using the scheduler in the different cloud types of resources under experiment so that the users can know which resources to use. Although we are experimenting with the scheduling algorithm in a cloud environment, the algorithm can be used in a grid computing setup as well. Our scheduling algorithm has the following advantages:

- Portability. Our scheduling algorithm and its implementation are not dependent on any specialized platform. The algorithm does not make any assumption about the underlying network topology and the underlying computing and networking capacity. Many of the works in scheduling assume a linear relationship between network and computing [20, 21] and are built on top of specialized platform, such as linear networks, single level trees, meshes and bus [20, 22].
- Considering network and computing latencies. Our algorithm computes dynamically the latencies before using their values into the formulas to compute the chunk sizes that should be allocated to the computing workers.
- Considering heterogeneous platform. Our scheduling algorithm considers a heterogeneous platform. Many of the works that were performed in scheduling algorithm are only applicable to homogeneous platforms [23]. We evaluate the performance of our technique using heterogeneous instances from the Amazon EC2. On the other hand, our scheme is generic and can be applied to homogeneous platforms as well.

One of the main objectives of our work is to obtain an optimal performance for an embarrassingly parallel application compared with its best *theoretical* and *ideal* measurements, and to obtain a speedup versus its sequential execution. The *theoretical* measurement is obtained by calculating the total makespan of the parallel application by using our formulas and the actual measured network power, computing power and their respective latency when running the application. The *ideal* measurement is obtained by considering a homogeneous platform and 0 network transfer time. We tested

DOI: 10.1002/spe

the implementation of our algorithm with three applications: a video compressor application, a portfolio management application calculating stock options prices and the ensemble clustering algorithm, which is used in data mining applications to analyze data sets. Significant performance can be obtained by scheduling the application's chunks considering the computing and network capabilities. Significant performance is obtained encouraging the use of our tool by other embarrassingly parallel applications to increase their performance in a cloud computing environment.

The rest of the paper is structured as follows. Section 2 describes the system model. Our dynamic scheduling algorithm is described in Section 3. Section 4 overviews related works. The implementation of our algorithm is described in Section 5. Section 6 describes the parallel applications we implemented using our scheduling approach. In Section 7, we evaluate the algorithm and discuss the obtained results. Section 8 concludes the work.

2. SYSTEM MODELING

As shown in Figure 1, a cloud consists of multiple physical clusters that are virtualized to users on a set of virtual clusters that are used to run users' applications. For divisible load applications, one need one master worker and multiple computing workers, which are independently connected to the master worker. Note that the data speed of the link connecting these computing workers, and the master worker dictates the speed of communication processing when a task is transmitted. The computing workers could be located physically at remote distance, and they are operating physically in an independent heterogeneous mode because each individual computer worker may consists of one or multiple virtual cores, which depends on the built-in capacity in terms of computing power, memory limitation and communication protocols of the shared physical resources.

Upon receiving of an application or a user's request, the master worker divides the whole application into a sequence of tasks, to be processed at the computing workers. Because each task involves certain amount of overheads for transmission and computing purposes, in order to minimize the overheads as well as to achieve an optimum performance efficiency, the master worker includes a scheduler, which must take into account the capacity of the selected computing workers in terms of the capacity of the communication link between the master worker and the selected computing worker, the available memory capacity and the existing computing power of the computing workers in the process of task partitioning, and scheduling and distribution. In this work, we assume the master sends the chunks to the workers in a synchronous way. This is a common assumption in the literature, which is justified by the network and input/output (I/O) bottleneck and scalability issues in data centers [24]; that is, the number of computing workers increase, the congestion on the Ethernet port of the master increases, and therefore, this could impose serious scalability issues.

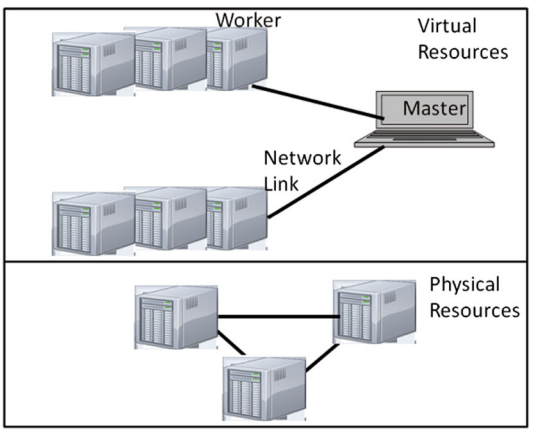

Figure 1. A cloud computing platform model.

DOI: 10.1002/spe

This paper focuses on the implementation and performance evaluation of a scheduling algorithm for divisible load applications in which tasks can be divided to run by multiple computing workers with no synchronization or communication. The parameter sweep applications fall into this category in which the same code is run multiple times using tasks or chunks of input parameter values (in the rest of the paper, we use the terms task and chunk interchangeably). The scheduler dynamically computes the network and computing capacities of the resources provided by the cloud for an application. Such feedback is a valuable information for the master to dynamically compute the chunks that should be distributed to the resources for an optimum performance efficiency. Note that the master worker can assign tasks to itself; that is, the master itself can be a computing worker. The master worker collects partial results from the computing workers and combines them to constitute the final application's result.

In the following analysis, a cloud consists of N heterogeneous computing workers. Each computing worker $i, , i \in \{1, \ldots, N\}$, in the cloud has a computing capacity, μ_i. An application consists of a total of W_{total} divisible computing loads. A portion of the total load, $chunk_i \leq W_{total}$, is processed by the computing worker i. We model the time required for a computing worker to perform the computation of $chunk_i$ units of load, TP_i, as follows. A unit of load is a unit of data input for an application. It could be one byte or several bytes, and it is an application dependent. For instance for a video, one unit of load can be defined as 1 s of video; for the Black–Scholes application, one unit of load is one option from a list of options; and for the Ensemble Clustering data mining application, one unit of load is one record of raw data.

$$TP_i = \theta_i + \frac{chunk_i}{\mu_i} \tag{1}$$

where θ_i, is a fixed latency, in seconds, for starting the computation at the computing worker i. μ_i is the computational speed of the worker in units of load per second.

We model the time required for sending $chunk_i$ units of load to computing worker i, TC_i, as follows:

$$TC_i = \lambda_i + \frac{chunk_i}{C_i} \tag{2}$$

where the component λ_i, in TC_i, is a latency, in seconds, incurred by the master to initiate data transfer to worker i. C_i represents the capacity of the communication link from the master to the worker i. It is a network characteristic; the data transfer rate in terms of bytes per second. However, as 1 unit of load is an indivisible unit of representation of a divisible load application, then C_i has to be converted, using a conversion factor, to units of load per second when using the formulas for computing the chunks to be distributed to the available computing workers.

Our extensive experiments show that depending on the application, there could be no linear relationship between the number of units of load (a chunk) and its corresponding size in bytes. Therefore, there is no linear relationship between the size of a chunk, represented in units of load and its transfer time over the network as in the formulas as suggested by Equation (2). More units of load can produce less data size in bytes, as it is the case for a video compressor application, such as the mencoder application, for example. Ignoring such a problem when applying the formulas of the scheduler could produce chunk sizes, which result in large discrepancies between the performance of the scheduler theoretical model and the real performance of the parallel application. Consequently, the scheduling algorithm will not be as efficient as it is assumed by the theoretical model. To solve this problem, for an application such as the mencoder, which has no constant conversion factor between the sizes of loads in bytes and their corresponding sizes in units of load, we compute the conversion factors of an important number of data samples, and we use the average conversion factor to convert C_i from bytes per second to units of load per second. Following this method, the percentage of degradation between the real performance of the application and its theoretical performance decreases considerably. For other applications, which represent a linear relationship between the number of units of load and the size in bytes and consequently, the transfer time over the network, such as the Black–Scholes and the K-Means applications, applying the scheduler increases the parallel application's efficiency and results in little discrepancies, which are

DOI: 10.1002/spe

Table I. Experimental values of network and computing latencies in local commodity clusters and geographically distributed clusters.

Master location	Computing worker location	Transfer latency: λ (seconds)	Computing latency: θ (seconds)
HPGCL-UAEU	Amazon EC2	10.092	10.071
Amazon EC2	Amazon EC2	0.793	1.344
HPGCL-UAEU	Cluster-UAEU	5.065	5.065
HPGCL-UAEU	HPGCL-UAEU	0.158	1.931
HPGCL-UAEU	NYU-Poly-USA	4.165	2.962

mostly due to the cloud environment virtualization and middleware. In this work, we pointed out the problem resulted from converting bytes to units of load when applying the theoretical model, and we suggested the use of a conversion factor to solve the problem.

The λ_i component in Equation (2) may be caused by the delay when initiating communicating from the master to the worker i by using SSH or any grid or cloud software, which is used to access the worker i. It represents the computing latency in seconds.

To have an idea of the magnitude and the impact of system latencies, both computing (θ_i) and networking (λ_i), on our scheduling performance, we measure those parameters in different setups, where workers could be located in a local commodity cluster or in a large-scale distributed cluster. To compute the computing latencies, we execute the mencoder on a remote machine using the *ssh* command with increasing size input files of videos, linear fitted those values and use the linear regression [25] to find the computing latency of 0-length of video. To measure the transfer latency, we use the *scp* command to transfer files with increasing sizes in bytes, linear fitted those values and use the linear regression to find the transfer latency of zero-length file. We measure the latencies in two scenarios in Amazon EC2: one in which the master worker is running within the Amazon EC2, and one in which the master worker is running in our HPGCL Research Laboratory at the United Arab Emirates University at UAE. Each measurement is repeated 25 times, and the average is taken. Table I shows that those latencies are not negligible, and they could be very high in case the master worker and the computing workers belong to different entities due to user authentication involved in the remote commands. In order to know the values of these latencies in commodity distributed systems, we measure the latencies within the UAE University and with distant machine located in the Polytechnic Institute of New York University in the USA. The results in Table I shows these values.

3. SCHEDULING ALGORITHM

To obtain an optimal performance, our main objective is to determine the chunk size that is to be assigned to each computing worker so that all computing workers complete their computations at the same time [21]. The total load of W_{total} computation units is divided into N computing workers. We assume that the master starts to send chunks to N available computing workers in a sequential fashion. Consequently, every computing worker i will have its chunk of size $chunk_i$ to process.

The time required by the first computing worker to complete the assigned task, $chunk_1$, is given by the following formula:

$$T_1 = TC_1 + TP_1 \tag{3}$$

$$= \lambda_1 + \frac{chunk_1}{C_1} + \theta_1 + \frac{chunk_1}{\mu_1} \tag{4}$$

The time required by the second computing worker to complete its task, $chunk_2$, is given by the following formula:

$$T_2 = TC_1 + TC_2 + TP_2 \tag{5}$$

For the first worker and the second one to complete their tasks at the same time, the following formula must hold:

DOI: 10.1002/spe

$$T_1 = T_2 \tag{6}$$

Based on Equations (1–3), (5) and (6), we obtain the following formula:

$$chunk_2 = \left[\frac{chunk_1}{\mu_1} - (\lambda_2 + \theta_2 - \theta_1)\right] \frac{C_2\mu_2}{C_2 + \mu_2} \tag{7}$$

Similarly, the time required by the third computing worker to complete its task, $chunk_3$, is given by the following formula:

$$T_3 = TC_1 + TC_2 + TC_3 + TP_3 \tag{8}$$

Consequently, for the second worker and the third one to complete their tasks at the same time, the following formula must hold:

$$T_2 = T_3 \tag{9}$$

Based on Equations (1), (2) and (7–9), we obtain the following formula to compute the chunk that should be allocated to worker 3, as follows:

$$\begin{aligned} chunk_3 = & \frac{chunk_1}{\mu_1} \frac{C_2C_3\mu_3}{(C_2 + \mu_2)(C_3 + \mu_3)} - (\lambda_2 + \theta_2 - \theta_1) \frac{C_2C_3\mu_3}{(C_2 + \mu_2)(C_3 + \mu_3)} \\ & - (\lambda_3 + \theta_3 - \theta_2) \frac{C_3\mu_3}{C_3 + \mu_3} \end{aligned} \tag{10}$$

By recursion, the value of $chunk_N$ is obtained by the following formula:

$$chunk_N = \frac{chunk_1}{\mu_1} \frac{\left(\prod_{i=2}^{i=N} C_i\right)\mu_N}{\prod_{i=2}^{i=N}(C_i + \mu_i)} - \sum_{i=2}^{i=N} \left[(\lambda_i + \theta_i - \theta_{i-1}) \frac{\left(\prod_{j=i}^{i=N} C_j\right)\mu_N}{\prod_{j=i}^{j=N}\left(C_j + \mu_j\right)}\right] \tag{11}$$

As $W_{total} = \sum_{i=1}^{i=N} chunk_i$ and based on the values $chunk_i$ found previously, then we obtain the value of $chunk_1$ as follows:

$$chunk_1 = \frac{W_{total} + \sum_{i=2}^{i=N}\sum_{j=2}^{j=i}\left(\lambda_j + \theta_j + \theta_{j-1}\right)\frac{\prod_{k=j}^{k=i}(C_k)\mu_i}{\prod_{k=j}^{k=i}(C_k+\mu_k)}}{1 + \sum_{i=2}^{i=N}\frac{\mu_i\prod_{j=2}^{j=i}C_j}{\mu_1\prod_{j=2}^{j=i}(C_j+\mu_j)}} \tag{12}$$

Based on Equation (12), the remaining chunks, $chunk_i$, $i = 2, ..., N$, can be obtained by substituting the value of $chunk_1$ in the corresponding equations.

However, the equations may not have a feasible solution, because some $chunk_i$ $(i = 2, ..., N)$ can be negative when the total load W is too small, and all the computing workers cannot take part of the computation. In that case, less computing workers should be used.

4. RELATED WORKS

The works in scheduling divisible load applications are divided into two categories. The first category consists of single-round algorithms in which a total workload is divided and distributed to a number of computing workers in one time [4]. The second category consists of multiround algorithms in which a total workload is divided and distributed to computing workers using several rounds in the purpose to overlap communication with computation and reduce the application's makespan. In both categories, some works consider a linear cost model in which there is a linear relationship between the computing time, the communication time and the data size. Other works assumes an affine cost model in which latencies are added to the linear cost model; which is considered more realistic [21]. In the first category, reference [4] introduces a scheduling algorithm for

DOI: 10.1002/spe

cluster of workstations. The algorithm considers an affine model on a star and bus network type. The experimental results on a compression application reveal from 10.5% to 30% of degradation compared with the theoretical, and the degradation was growing with increasing workload on IBM SP2 platform, while decreasing from 55% to 7% with increasing workload on a homogeneous PCs-based platform. The multi-installment algorithm, studied in [20], proposed a model of scheduling tasks in a single round, but communication time and computation time are assumed to be proportional, and latencies are not considered. In our previous work [26], we modeled the system usage of a single-round scheduling, but we assumed fixed latencies and did not practice and experience with real applications in a real environment. In reference [27], we introduced a scheduler for the conjugate gradient application in a cloud computing environment, but we did not consider dynamic network capacities. Other works were performed on specialized hardware, such as in [22] for a distributed bus network.

In the second category of works on scheduling algorithms, a multiround algorithm [17] was built [20] by adding communication and computation latencies to the model. The algorithm aims at optimizing the number of rounds to obtain an optimal makespan of the application. Yang *et al.* [28] studies the complexity of multiround divisible load scheduling and concluded that the combination of the selection process of computing workers, their ordering and the distribution of the adequate chunk sizes is NP-complete. Drozdowski and Lawenda [29] built up a scheduling model with the assumptions that computations are suspended by communications.

In our current work, we focus on experimenting and evaluating the performance of a single-round scheduler in the Amazon EC2 environment. Our extensive experiments on the Amazon EC2 platform reveal the performance of the scheduler in the cloud and the types of instances that should be used for better performance. We learned that a scheduler becomes inefficient if the conversion between bytes and units of load is not taken care of during implementation.

5. SCHEDULER IMPLEMENTATION

The scheduler implementation uses the formulas described in Section 3 to compute the chunks that must be distributed to computing workers. These formulas must take an input the values of μ (computing power) and θ (computing latency) for each computing worker, as well as the values of C (network bandwidth) and λ (network latency) for the communication links between the master worker and each of the computing workers. The scheduler computes these parameters by using linear regression [25]. The scheduler then computes the chunks according to our derived formulas. The values of the computed chunks are then used by the application master worker in order to divide the application accordingly and allocate the application's chunks to the computing workers, by decreasing order of the value of C; that is, it allocates to the first computing worker, whose communication link C is the biggest (i.e., the fastest), its corresponding chunk. After sending the computation chunk to its corresponding computing worker, the master worker invokes the remote execution on the computing worker by using the *ssh* command. We configure a passwordless communication between the master worker and each of the computing workers. In our implementation, each computing worker sends a signal to the master worker upon its completion.

6. EXPERIMENTED APPLICATIONS

6.1. Mencoder

The mencoder is a video compressor application. We use the mencoder source code version 4.45, which is included in the Mplayer project [30]. We use the MPEG-4 (or H264) video format [31], with resolution of 1920×1080 of video input, 24 bpp, 23 fps and 18,356.7 kbps (2240.8 kbyte/s). In mencoder, a unit of load is expressed by seconds of video. A master worker reads the video input from a file and divides it and distributes it to the computing workers for parallel compression of the video parts. Each computing worker compresses its part of the video and sends the result to the master worker. The results can then be used to be encoded into another format. We modified the master worker code to incorporate our scheduler.

DOI: 10.1002/spe

6.2. Portfolio management: Balck–Scholes model

We use the portfolio management application source code [32], which is part of the Princeton Application Repository for Shared-Memory Computers benchmark suite [33]. The application uses the Black–Scholes formula, which is a widely used formula to calculate the prices for a portfolio of European options using the Black–Scholes model. The input data file of the application stores a number of total options, followed by derivatives in an array data structure. The application iterates over the input file. At every iteration, the application reads an option from the file, applies the Black–Scholes formula on it, obtains the calculated prices as an output and write the price in an output file.

6.3. Data mining: ensemble clustering

Data streams are continuous flows of data. Examples of data streams include network traffic, sensor data, call center records and so on. Their sheer volume and speed pose a great challenge for the data mining community to mine them [10]. Multistep methodologies and techniques, and multiscan algorithms, suitable for knowledge discovery and data mining, cannot be readily applied to data streams. This is due to well-known limitations, such as bounded memory, high speed data arrival, online/timely data processing and need for one-pass techniques (i.e., forgotten raw data), issues and so on. For this, we have studied ensemble-based techniques to classify stream data [10, 34]. Here, a set of model in the ensemble is maintained. A model is generated using clusters to represent finer gain decision boundary. Cluster will summarize information from raw data and keep only summary information by discarding raw data. However, the clustering algorithm (k-Means) may hinder the performance of online classification when a single machine is used. Therefore, cloud-based solution is used here to build a number of clusters from a large number of raw data to represent a model.

We use the forest cover (UCI Repository – http://archive.ics.uci.edu/ml/): The data set contains geospatial descriptions of different types of forests. It contains seven classes, 54 attributes and around 581,000 instances.

7. PERFORMANCE EVALUATION

In this section, we evaluate the efficiency of our scheduling algorithm by implementing it and experimenting with three applications: a video compressor application, a portfolio management application based on the Black–Scholes model and a data mining application based on the k-Means clustering. The efficiency of the algorithm is measured and compared with the *ideal* makespan and to the *theoretical* one. The *ideal* makespan is calculated by dividing the total workload over the summation of the computing powers of the individual workers $\left(\frac{W_{total}}{\sum_{i=1}^{i=N}\mu_i}, i = 1, ..., N\right)$. The theoretical one is the one obtained theoretically by applying the formulas obtained in this study. In particular, we analyze the impact of the system parameters (μ_i, C_i, θ_i, and λ_i) in a cloud environment, such as the Amazon EC2 environment, on the performance of the applications using our scheduling algorithm. The experiments also reveal the impact of the cloud infrastructures chosen on the performance, as well as the efficiency of the scheduling algorithm.

7.1. Experimental environment

The experiments use three types of virtual clusters from Amazon EC2 environment. The choice of these clusters is motivated by their reasonable price, assumed to be widely used by the majority of public. Each cluster has a set of 10 computing workers and 1 master worker. The clusters are as follows: (1) A cluster made of Small Standard instances with 1.7 GB memory, 1 EC2 Compute Unit. 1 EC2 Compute Unit [35] is equivalent to a CPU capacity of 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor. The I/O performance is defined to be moderate. (2) A cluster made of Large Standard instance with 7.5 GB memory, 4 EC2 of computing units, and the I/O performance is defined to be high; and (3) a heterogeneous cluster made of four instances of Small Standard, three instances of Large Standard and three instances of High-Memory Extra. A Cluster High-Memory Extra Large instance provides 1.7 GB of memory, 6.5 EC2 computing units, and the I/O performance is defined

DOI: 10.1002/spe

to be moderate. For the heterogeneous cluster, the master worker runs on a Large instance. The choice of these instances types for the heterogeneous cluster is motivated by the objective to obtain a heterogeneity in computation to communication ratio between the master worker and each of the computing workers. We created all the instances in Virginia, US East. In all our experiments, the master worker and the computing workers are running in the Amazon Cloud computing infrastructure. We believe that users would be more interested to run the master worker in the cloud rather than on a laptop or a local machine to have the freedom of launching their application in the cloud, disconnecting and reconnecting later to obtain the results. In the case of the heterogeneous cluster, the master runs in the Standard Large instance type. In all our experiments, we ordered the computing workers by decreasing order of network power [36]. Although a better performance could be obtained by using different ordering heuristics, this is not the focus of our work here.

For each experimental environment, we compute the network bandwidth (C) and the network latency (λ) by using a linear regression[25] at large-size-data fit and at 0-fit, respectively between the master and each of the computing workers. The computing power of each computing worker (μ) and its corresponding latency (θ) are computed for each application by executing the application using units of loads. We also use the linear regression at large-size-data fit for μ and at 0-fit for θ. Tables II–IV show the infrastructure characteristics for the Small Standard, Large Standard, and heterogeneous virtual clusters, respectively. Those values are computed once at the clusters creation.

7.2. Experiments

The total execution times for the parallel applications under experiment are measured. The total execution time for a parallel application is the elapsed time when master starts distributing the chunks till the master receives all the signals from all the computing workers that execution has completed. The idle time of a computing worker is the time spent by a computing worker waiting to receive its chunk. Note that the idle time of the worker does not include the communication time between the master and the worker. In order to asses the impact of increasing workload on the efficiency of the scheduling algorithm, we measure the total execution time of the application with increasing input data size. We calculate the relative performance efficiency of the application using the scheduler and the cloud computing environment by computing the relative makespan of the actual measurement versus the *ideal* makespan. We also calculate the percentage of degradation from the theoretical ((($actual\text{-}theoretical$)$/theoretical$) $*$ 100), which indicates to which extent the theoretical model holds in the environments under experiments. For the homogeneous virtual clusters, we also compute the speedup obtained by the applications under experiment when running with the scheduler. In all our experiments, we perform each run 100 times, and the average is computed.

7.3. Experimental results analysis

In deriving the scheduling algorithm for divisible load applications, the following requirements are considered. Those requirements have an effect on the experimental results obtained, as follows:

- Portability: Portability is required in twofold:
  - The cloud computing presents virtualized resources to users. Consequently, the scheduling algorithm should not be dependent on any specialized hardware or propriety software. Our implementation does not have hardware or software dependencies.
  - Divisible load applications are very common. Our experiments show that it is easy to integrate our scheduler in existing applications to increase their performance.
- Considering latencies: A cloud computing present to users resources with latencies. In our implementation, the scheduler computes these latencies, which are then considered in the computation of chunks to increase the overall application performance efficiency.
- Considering heterogeneous environments: One of our main objectives is to consider heterogeneity in allocating chunks to resources. The computation of the computing powers and the network capacities of a cluster are implemented so that heterogeneity is considered.

Upon implementing the scheduler, our first aim is to find how much the actual measured application's total makespan is close to the theoretical one. Figures 2–4 show the percentage of degradation

DOI: 10.1002/spe

Table II. Experimental values of system parameters using Standard Small instance type in Amazon Elastic Cloud Computing.

Node name	C (bytes/sec)	λ (seconds)	μ (units of load/s) for mencoder	θ (seconds) for mencoder	μ for Black–Scholes	θ for Black–Scholes	μ for BK-Means	θ for K-Means
Node-1	18,473,500	0.60	0.32	0.67	11,345.50	0.72	980.76	2.76
Node-2	18,046,200	0.57	0.31	0.67	11,043.50	0.75	963.12	2.85
Node-3	18,457,500	0.61	0.32	0.67	11,316.70	0.72	978.10	2.77
Node-4	17,909,200	0.78	0.32	0.67	11,322.30	0.67	987.87	2.72
Node-5	17,939,500	0.54	0.31	0.67	11,176.50	0.66	969.79	2.76
Node-6	18,153,500	0.46	0.32	0.67	11,266.70	0.70	986.35	2.67
Node-7	18,651,500	0.61	0.32	0.67	11,304.70	0.70	980.87	2.86
Node-8	18,033,700	0.59	0.32	0.67	11,265.20	0.70	973.50	2.65
Node-9	18,135,100	0.54	0.32	0.67	11,322.30	0.70	973.88	2.79
Node-10	18,736,100	0.59	0.32	0.67	11,382.80	0.65	983.63	2.74

DOI: 10.1002/spe

Table III. Experimental values of system parameters using Standard Large instance type in Amazon Elastic Cloud Computing.

Node name	C (bytes/s)	λ (seconds)	μ (units of load/s) for mencoder	θ (seconds) for mencoder	μ for Black–Scholes	θ for Black–Scholes	μ for BK-Means	θ for K-Means
Node-1	34,488,800	0.49	0.49	0.60	19,954.70	0.49	1505.60	1.59
Node-2	36,512,600	0.43	0.50	0.60	20,213.40	0.37	1568.80	1.30
Node-3	36,165,600	0.39	0.50	0.60	20,214.30	0.40	1584.76	1.31
Node-4	37,263,600	0.43	0.50	0.60	20,253.70	0.44	1630.83	1.41
Node-5	35,177,600	0.44	0.51	0.60	20,214.30	0.40	1610.28	1.36
Node-6	36,527,800	0.43	0.50	0.60	20,252.80	0.38	1566.05	1.30
Node-7	41,467,900	0.43	0.57	0.60	23,832.20	0.34	1777.28	1.14
Node-8	36,847,300	0.44	0.50	0.60	20,214.30	0.40	1610.24	1.21
Node-9	35,338,500	0.45	0.49	0.60	20,017.80	0.43	1514.09	1.43
Node-10	36,640,200	0.44	0.50	0.60	20,231.20	0.38	1619.49	1.29

DOI: 10.1002/spe

Table IV. Experimental values of system parameters using heterogeneous resources from Amazon Elastic Cloud Computing.

Node name	C (bytes/s)	λ (seconds)	μ (units of load/s) for mencoder	θ (seconds) for mencoder	μ for Black–Scholes	θ for Black–Scholes	μ for BK-Means	θ for K-Means
Node-1	18,551,100	0.52	0.32	0.67	11,352.70	0.67	972.27	2.80
Node-2	19,297,900	0.52	0.29	0.67	11,886.20	0.73	883.10	2.95
Node-3	18,961,700	0.47	0.29	1.33	11,901.10	0.75	870.44	2.99
Node-4	19,050,000	0.34	0.29	0.67	11,910.60	0.70	895.07	2.98
Node-5	39,873,800	0.44	0.68	0.33	24,870.70	0.38	2029.44	1.07
Node-6	36,523,600	0.40	0.50	1	20,244.90	0.42	1561.05	1.27
Node-7	35,401,000	0.40	0.50	0.33	20,138.70	0.43	1577.07	1.33
Node-8	39,181,200	0.35	0.85	0.67	30,336.00	0.37	2712.28	0.89
Node-9	39,088,700	0.31	0.88	0.33	30,290.40	0.37	2639.90	0.85
Node-10	38,603,300	0.37	0.85	0.33	30,122.90	0.29	2663.43	0.98

DOI: 10.1002/spe

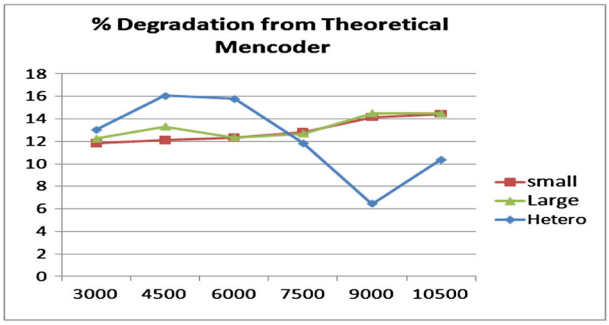

Figure 2. Percentage of degradation from the theoretical best for the mencoder application.

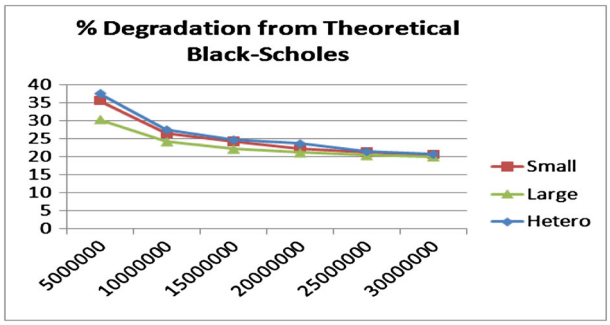

Figure 3. Percentage of degradation from the theoretical best for the portfolio management application.

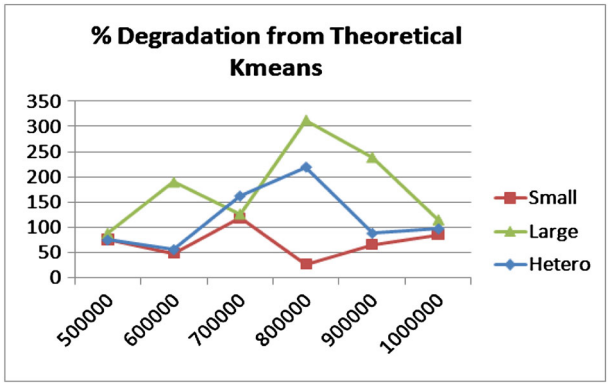

Figure 4. Percentage of degradation from the theoretical best for the data mining application.

of the measured total makespan of those applications from the theoretical calculated makespan. The theoretical makespan is calculated on the basis of the actual measured communication and computation latencies which we use as input to our derived formulas. For the mencoder application, the percentage of degradation is generally decreasing with increasing workload in the heterogeneous from 13.03% for 3000 s of video to 6.45% for 9000 s of video. The theoretical model is holding better for smaller data sizes than for larger ones in the small and large clusters, due to the larger communication to computation ratio in the mencoder application for large data set than for smaller data set. However, a speedup of 5.7 and 7.9 are still obtained in small and large clusters, respectively for the large data size of 10,500 s of video as shown in Figure 5. The efficiency of the scheduling algorithm and the impact of cloud infrastructure are also shown in Figure 6 where the gap between the different infrastructure is due to overhead in communication time spent between the master worker and the computing workers.

DOI: 10.1002/spe

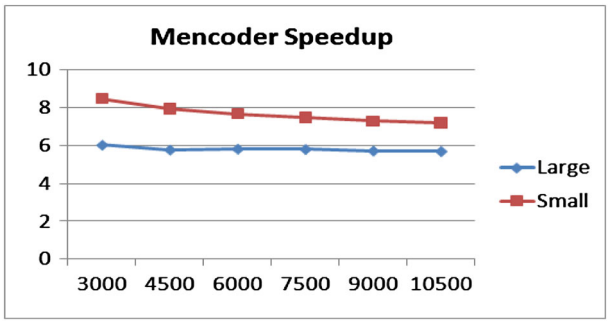

Figure 5. Speedup of the parallel mencoder application compared with its sequential execution.

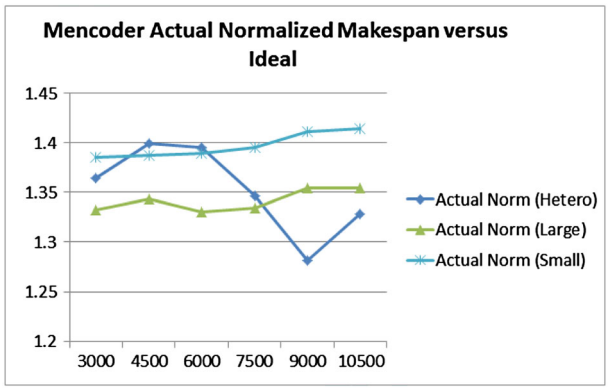

Figure 6. Relative makespan compared with the ideal makespan for the mencoder application.

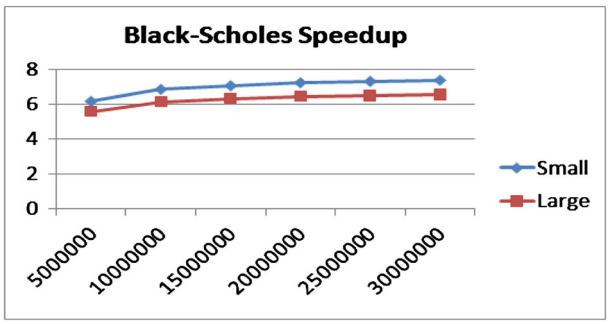

Figure 7. Speedup of the parallel portfolio management application compared with its sequential execution.

Figure 3 shows that degradation of the actual makepsan compared with the theoretical one decreases with increasing workload for the Black–Scholes application; about 20% for large data set for all the infrastructures under experiment. The speedup of the application also increases with increasing workload to reach 7.39 for the small cluster and 6.58 for the large cluster with a large data set as shown in Figure 7. Figure 8 shows that the actual makespan becomes closer to the ideal one with increasing workload as the application is mostly computing intensive, and the computation to communication ratio is higher than that of the mencoder application, for example.

For the K-Means application, our experiments show that the behavior of the application is such that there is no linear relationship between a data set size and its execution time on a computing worker. This is an interesting observation, which participates to an increase in degradation of the actual makespan compared with the theoretical one as shown in Figure 4. The scheduler assumes a linear relationship between the size of the data set in units of load and its execution time. This does not hold for the K-Means application, and therefore, all the computing workers do not complete at

DOI: 10.1002/spe

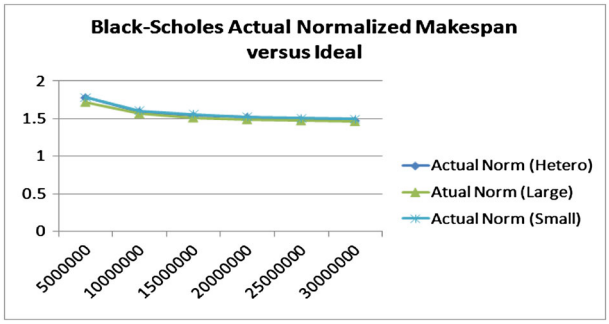

Figure 8. Relative makespan compared with the ideal makespan for the portfolio management application.

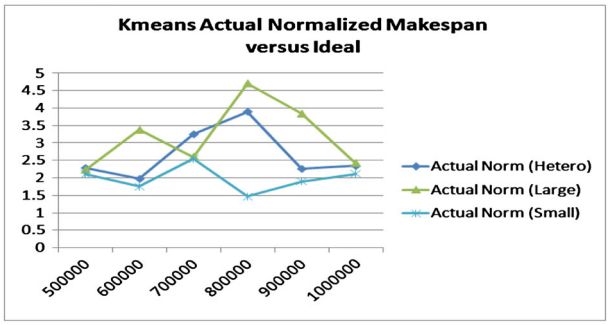

Figure 9. Relative makespan compared with the ideal makespan for the data mining application.

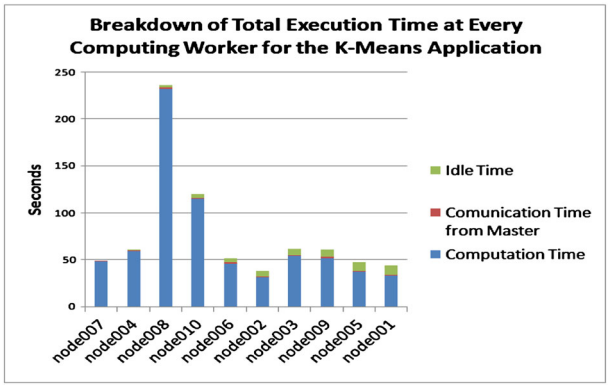

Figure 10. Total execution time at each computing worker for the data mining parallel application in the large cluster.

the same time actually, participating in an increase of the system idle time and consequently the total makespan (see also Figure 9). Figure 10 shows the breakdown of execution on every computing worker in the large cluster for the K-Means application, and we can see that all the computing workers do not actually complete their execution at the same time as assumed in the theoretical model. For the K-Means application, a maximum speedup of 10.7 was obtained for a large data of 900,000 units of load on the large cluster. And a speedup of 14.36 was obtained on the small cluster for 600,000 units of load.

8. CONCLUSION

Divisible load applications are very common in scientific and engineering applications. They are of type embarrassingly parallel applications or parameter sweep applications in which a global

DOI: 10.1002/spe

load can be divided into chunks that can be performed separately by parallel computing workers. There is no synchronization or communication between the workers. In those applications, a master worker distributes the load to the computing workers which perform the load in parallel and return their results to the master. Scheduling is one of the techniques to increase the performance efficiency of divisible load applications on a heterogeneous platform. Cloud computing is seen as an effective infrastructure for high performance applications to reduce the cost of running those applications on commodity dedicated clusters. Cloud computing, however, incurs overhead of virtualization and cloud middleware. It presents to users virtualized resources. So it is an open question whether a scheduling can result in application performance efficiency. In this work, we implemented and evaluated the performance of a scheduling algorithm that takes into consideration the network and computational power of the virtual resources as well as the corresponding latencies involved when running the application. We evaluate the performance with real applications and using a real and very well-known cloud computing infrastructure, such as the Amazon EC2 infrastructure. The results show that an important application speedup, and efficiency can be obtained using our scheduling algorithm. We obtained a ratio of 1.4 and 1.78 maximum of relative performance of actual makespan compared with the *ideal* makespan for the mencoder and the portfolio management applications and a ratio of 4.7 for the ensemble clustering application. The scheduling algorithm becomes even more interesting for applications which present a linear relationship between the number of units of load (chunk size) and its computation time such as mencoder and portfolio management application, and for applications which present a linear relationship between the number of units of load and its transfer time over the network, such as in the portfolio management application. All the parallel applications under study, using the scheduler and the cloud computing infrastructure, presented a speedup compared with its sequential execution. Our work continues to evaluate the performance of multiround scheduling algorithms for parallel applications in a cloud environment. This is to reveal the impact of the choice of both the platform and the algorithm on the applications' performance.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Kang Xi from the Polytechnic Institute of New York University in the USA for granting access to their machines in the USA to perform latencies experiments between the UAE and the USA.

REFERENCES

1. Bharadwaj V, Ghose D, Robertazzi T. Divisible load theory: a new paradigm for load scheduling in distributed systems. *Cluster Computing* 2003; **6**(1):7–17.
2. Ghose D, Robertazzi T, editors. Special issue on Divisible Load Scheduling. *Cluster Computing* 2003; **6**(1).
3. Robertazzi T. Ten reasons to use divisible load theory. *IEEE Computer* 2003; **36**(5):63–68.
4. Drozdowski M, Wolniewicz P. Experiments with scheduling divisible tasks in cluster of workstations. *Euro-Par 2000*, Munich, Germany, 2000; 311–319.
5. Altilar D, Paker Y. An optimal scheduling algorithm for parallel video processing. *In IEEE International Conference on Multimedia Computing and Systems*, Austin, TX, IEEE Computer Society Press, 1998.
6. Altilar D, Paker Y. Optimal scheduling algorithms for communication constrained parallel processing. *In Euro-Par 2002, LNCS 2400*, Munich, Germany, Springer Verlag, 2002; 197–206.
7. Barlas G. Cluster-based optimized parallel video transcoding. *Parallel Computing* 2012; **38**:226–244.
8. Lee C, Hamdi M. Parallel image processing applications on a network of workstations. *Parallel Computing* 1995; **21**:137–160.
9. Bharadwaj V, Ranganath S. Theoretical and experimental study on large size image processing applications using divisible load paradigms on distributed bus networks. *Image and Vision Computing* 2002; **20**(13–14):917–1034.
10. Masud MM, Gao J, Khan L, Han J, Thuraisingham B. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering (TKDE), 2011, IEEE Computer Society* 2011; **23**(6):859–874.
11. Buyya R, Yeo CS, Venugopal S. Market-oriented cloud computing: vision, hype, and reality for delivering it services as computing utilities, keynote paper. *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC 2008)*, IEEE CS Press, Dalian, China, Sept. 25Ű27, 2008.
12. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 2009; **25**(6).

DOI: 10.1002/spe

13. Armbrust M, Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. Above the clouds: a Berkeley view of cloud computing. *Technical Report No. UCB/EECS- 2009-28*, University of California at Berkley, USA, February 10, 2009.
14. Vecchiola1 C, Pandey1 S, Buyya R. High-performance cloud computing: a view of scientific applications. *Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN 2009, IEEE CS Press, USA)*, Kaohsiung, Taiwan, Fall December 14.
15. Service Level Agreement Zone. The Service Level Agreement, @Copyright 2007. Available from: http://www.sla-zone.co.uk/index.htm [last accessed 04 June 2012].
16. Ismail L, Zhang L, Shuaib K, Bataineh S. Performance evaluation of a dynamic single round scheduling algorithm for divisible load applications. *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2012)*, Las Vegas, Nevada, USA, 2012.
17. Yang Y, van der Raadt K, Casanova H. Multiround algorithms for scheduling divisible loads. *IEEE Transactions on Parallel and Distributed Systems* November 2005; **16**(11).
18. Shokripour A, Othman M, Ibrahim H, Subramaniam S. New method for scheduling heterogeneous multi-installment systems. *Future Generation Computer Systems* 2012; **28**(8):1205-1216.
19. Available from: http://aws.amazon.com/ec2/ [last accessed 04 June 2012].
20. Bharadwaj V, Ghose D, Mani V. Multi-installment load distribution in tree networks with delays. *IEEE Transactions Aerospace and Electronics Systems* 1995; **31**(2):555–567.
21. Beaumount O, Casanova H, Legrand A, Robert Y, Yang Y. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Transactions on Parallel and Distributed Systems* 2005; **16**(3):207–218.
22. Bataineh S, Hsiung T-Y, Robertazzi TG. Closed form solutions for bus and tree networks of processors load sharing a divisible job. *IEEE Transactions on Computers* 1994; **43**(10):1184–1196.
23. Yang Y, Casanova H. Extensions to the multi-installment algorithm: affine costs and output data transfers. *Technical Report CS2003-0754*, Dept. of Computer Science and Engineering, University of California, San Diego, 2003.
24. Greenberg A, Hamilton JR, Jain N, Kandula S, Kim C, Lahiri P, Maltz DA, Patel P, Sengupta S. VL2: a scalable and flexible data center network. *SIGCOMM '09 Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ACM New York, NY, USA, 2009; 51–62.
25. Available from: http://www.statisticshowto.com/articles/how-to-find-a-linear-regression-equation/ [last accessed 15 September 2013].
26. Ismail L, Mills B, Hennebelle A. A formal model of dynamic resource allocation in grid computing environment. *Proceedings of The IEEE Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD2008)*, Phuket, Thailand, 2008; 685–693.
27. Ismail L, Barua R. Implementation and performance evaluation of a distributed conjugate gradient method in a cloud computing environment. *Software: Practice and Experience* 2013; **43**(3):281–304.
28. Yang Y, Casanova H, Drozdowski M, Lawenda M, Legran A. On the complexity of the multi-round divisible load scheduling, ISSN0249-6399, INRIA, 2007.
29. Drozdowski M, Lawenda M. Multi-installment divisible load processing in heterogeneous systems with limited memory. *PPAM 2005, LNCS 3911*, Poznan, Poland, 2006; 847–854.
30. Source code Available at: http://www.mplayerhq.hu/design7/dload.html [last accessed 15 September 2013].
31. Available from: http://www.h264info.com/h264.htmlis [last accessed 15 September 2013].
32. PARSEC. Available from: http://parsec.cs.princeton.edu/download.htm [last accessed 15 September 2013].
33. Biania C. Benchmarking modern multiprocessors. *PhD Dissertation*, Princeton University, 2011. Available from: http://parsec.cs.princeton.edu/publications/bienia11benchmarking.pdf [last accessed 15 September 2013].
34. Masud MM, Gao J, Khan L, Han J, Hamlen KW, Oza NC. Facing the reality of data stream classification: coping with scarcity of labeled data. *International Journal of Knowledge and Information Systems (KAIS)* 2012; **33**:2130244. Springer.
35. Amazon. What is a EC2 Compute Unit and why did you introduce it? Available from: http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it [last accessed 15 September 2013].
36. Beaumont O, Carter L, Ferrante J, Legrand A, Robert Y. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. *Proceedings Int'l Parallel and Distributed Processing Symposium (IPDPS)*, Marriott Marina, Fort Lauderdale, Florida, USA, June 2002.

DOI: 10.1002/spe