Performance Evaluation of Infrastructure as Service Clouds with SLA Constraints

Anuar Lezama Barquet[1], Andrei Tchernykh[1], and Ramin Yahyapour[2]

[1] Computer Science Department, CICESE Research Center, Ensenada, BC, Mexico

[2] GWDG – University of Göttingen, 37077 Göttingen, Germany

{alezama, chernykh}@cicese.edu.mx, ramin.yahyapour@gwdg.de

Abstract. In this paper, we present an experimental study of job scheduling algorithms in infrastructure as a service type in clouds. We analyze different system service levels which are distinguished by the amount of computing power a customer is guaranteed to receive within a time frame and a price for a processing time unit. We analyze different scenarios for this model. These scenarios combine a single service level with single and parallel machines. We apply our algorithms in the context of executing real workload traces available to HPC community. In order to provide performance comparison, we make a joint analysis of several metrics. A case study is given.

Keywords. Cloud computing, infrastructure as a service, quality of service, scheduling.

Evaluación del desempeño de servicios de infraestructura en nubes con restricciones de acuerdos de nivel de servicio (SLA)

Resumen. En el presente artículo, mostramos un estudio experimental sobre algoritmos de calendarización en servicios de infraestructura en nubes. Analizamos diferentes niveles de servicios que se distinguen por la cantidad de poder computacional que al usuario se le garantiza recibir dentro de un periodo de tiempo y el precio por unidad de procesamiento. Analizamos diferentes escenarios para este modelo. Estos escenarios combinan un único nivel de servicio en una sola máquina y en máquinas paralelas. Utilizamos nuestros algoritmos para la ejecución de muestras de cargas de trabajo reales disponibles para la comunidad de HPC. Con el fin de proveer una comparación en el desempeño, realizamos un análisis conjunto de varias métricas. Presentamos un caso de estudio.

Palabras clave. Computación en nube, servicio de infraestructura en nube, calidad de servicio, calendarización.

1 Introduction

Infrastructure as a service type in clouds allows users to take advantage of computational power on-demand. The focus of this kind of clouds manages virtual machines (VMs) created by users to execute their jobs on the cloud resources. However, in this new paradigm, there are issues that prevent its widespread adoption. The main concern is its necessity to provide Quality of Service (QoS) guarantees [1].

The use of Service Level Agreements (SLAs) is a fundamentally new approach for job scheduling. In this approach, schedulers are based on satisfying QoS constraints. The main idea is to provide different levels of service, each addressing a different set of customers for the same services, in the same SLA, and establish bilateral agreements between a service provider and a service consumer to guarantee job delivery time depending on the selected level of service. Basically, SLAs contain information such as the latest finish time of the job, reserved time for job execution, number of CPUs required, and price per time unit.

The shifting emphasis of the Grid and Clouds towards a service-oriented paradigm led to the adoption of SLA as a very important concept, but at the same time led to the problem of finding the stringent SLAs.

There has been significant amount of research on various topics related to SLAs: admission control techniques [2]; incorporation of the SLA into the Grid/Cloud architecture [3]; specifications of SLAs [4, 5]; usage of SLAs for resource management; SLA-based scheduling [6], SLA profits [7]; automatic negotiation protocols [8]; economic aspects associated with the usage of SLAs for service provision [9], etc. Little is known about the worst case efficiency of SLA scheduling solutions. There are only very few theoretical results on SLA scheduling, and most of them address real time scheduling with given deadlines.

Baruah and Haritsa [10] discuss the online scheduling of sequential independent jobs on real time systems. They presented the algorithm ROBUST (Resistance to Overload By Using Slack Time) which guarantees a minimum slack factor for every task. The slack factor *f* of a task is defined as a ratio of its relative deadline to its execution time requirement. It is a quantitative indicator of the tightness of the task deadline. The algorithm provides an effective processor utilization (EPU) of (*f*-1)/*f* during the overload interval. He shows that given enough processors, on-line scheduling algorithms can be designed with performance guarantees arbitrarily close to that of optimal uniprocessor scheduling algorithms.

A more complete study is presented in [11] by Schwiegelshohn *et al*. The authors theoretically analyze the single (SM) and the parallel machine (PM) models subject to jobs with single (SSL) and multiple service levels (MSL). Their analysis is based on the competitive factor which is measured as the ratio of the income of the infrastructure provider obtained via the scheduling algorithm to the optimal income. They provide worst case performance bounds of four greedy acceptance algorithms named SSL-SM, SSL-PM, MSL-SM, MSL-PM, and two restricted acceptance algorithms MSL-SM-R, and MSL-PM-R. All of them are based on adaptation of the preemptive EDD (Earliest Due Date) algorithm for scheduling jobs with deadlines.

In this paper, we make use of IaaS cloud model proposed in [11]. To show practicability and competitiveness of the algorithms, we conduct a comprehensive study of their performance and derivatives using simulation. We take into account an important issue that is critical for practical adoption of the scheduling algorithms: we use workloads based on real production traces of heterogeneous HPC systems.

We study two greedy algorithms: SSL-SM and SSL-PM. SSL-SM accepts every new job for a single machine if this job and all previously accepted jobs can be completed in time. SSL-PM accepts jobs considering all available processors in parallel machines. Key properties of SLA should be observed to provide benefits for real installations. Since SLAs are often considered as successors of service oriented real time paradigm with deadlines, we start with a simple model with a single service level on a single computer, and extend it to a single SLA on multiple computers.

One of the most basic models of SLA provides relative deadline as a function of the job execution time with a constant service level parameter of usage. This model does not match every real SLA, but the assumptions are nonetheless reasonable. It is still a valid basic abstraction of SLAs that can be formalized and automatically treated.

We address an online scheduling problem. The jobs arrive one by one and after the arrival of a new job the decision maker must resolve whether he rejects this incoming job or schedules it on one of the machines. The problem is online because the decision maker has to resolve it without information about the following jobs. For this problem, we measure the performance of the algorithms by a set of metrics which includes the competitive factor and the number of accepted jobs.

2 Scheduling Model

2.1 Formal Definition

In this work, we consider the following model. A user submits jobs to a service provider, which has to guarantee some level of service (*SL*). Let $S=[S_1,S_2,...,S_i,...,S_k]$ be the set of service levels offered by the provider. For a given service level S_i the user is charged at a cost u_i per unit of

execution time depending on the urgency of the submitted job. $u_{max} = \max_i\{u_i\}$ denotes the maximum cost.

The urgency of the job is denoted by the slack factor $f_i \geq 1$. The total number of jobs submitted to the system is n_r. Each job J_j from the released job set $J_r = [J_i, J_2, \ldots, J_{n_r}]$ is described by a tuple (r_j, p_j, S_i, d_j): its release date $r_j \geq 0$, its execution time p_j, and the *SL* S_i. The deadline of each job d_j is calculated at the release of the job as $d_j = r_j + f_i \cdot p_j$. The maximum deadline is denoted by $d_{max} = \max_j\{d_j\}$. The processing time of the job p_j becomes known at time r_j. Once the job is released, the provider has to decide, before any other job arrives, whether the job is accepted or not. In order to accept the job J_j the provider should ensure that some machine in the system is capable of completing J_j before its deadline. In the case of acceptance, further jobs should prevent that the job J_j misses its deadline. Once a job is accepted, the scheduler uses some heuristic to schedule the job. Finally, the set of accepted jobs $J = [J_1, J_2, \ldots, J_n]$ is a subset of J_r where n is the number of jobs successfully accepted and executed.

2.2 Metrics

We used several metrics to evaluate the performance of our scheduling algorithms and SLAs. In contrast to traditional scheduling problems, the classic scheduling metrics such as C_{max} become irrelevant in evaluating the system performance of systems scheduled through SLAs.

One of the objective functions represents the goal of the infrastructure provider who wants to maximize his total income. Job J_j with service level S_i generates income $u_i \times p_j$ in the case of acceptance and zero otherwise. The competitive factor $c_v = \dfrac{\sum_{j=1}^{n} u_i \times p_j}{\mathrm{V}(\mathrm{A})^*} \leq 1$ is defined as a ratio of total income generated by an algorithm to optimal income $\mathrm{V}(\mathrm{A})^*$. Due to maximization of income, a larger competitive factor is better than a smaller one. Note that in our evaluation of experiments, we use the upper bound of the optimal income $\hat{\mathrm{V}}(\mathrm{A})^*$ instead of the optimal income as we are, in general, not able to determine the optimal income.

$$\mathrm{V}(\mathrm{A})^* \geq \hat{\mathrm{V}}(\mathrm{A})^* = \min(u_{max} \cdot \sum_{j=1}^{n_r} p_j, u_{max} \cdot d_{max} \cdot m)$$

The first bound is the sum of the processing times of all released jobs multiplied by the maximum price per unit execution of all available SLAs. The second bound is the maximum deadline of all released jobs multiplied by the maximum price per unit execution value and the number of machines in the system. Due to our admission control policy, the system does not execute jobs whose deadline cannot be reached; therefore, this second bound is also an upper bound of the maximum processing time in which the system can execute work.

In our experiments we analyze SSL-SM and SSL-PM algorithms, since only one SL is used; we do not take u_{max} into account to calculate the competitive factor. We also calculate the number of rejected jobs and use it as a measure of the capacity of the system to respond to the incoming flow of jobs. Finally, we calculate the mean waiting time of the jobs within the system as $\mathrm{MWT} = \dfrac{1}{n}\sum_{j=1}^{n}(c_j - p_j)$, where c_j is the completion time of the job j.

3 Experimental Setup

3.1 Algorithms

In our experiments, we use SSL-SM and SSL-PM algorithms based on the EDD (Earliest Deadline Deadline) algorithm, which gives priority to jobs according to their deadline. The jobs that have been admitted but not yet completed are introduced in a queue. The jobs are ordered in non-decreasing deadlines. For their execution,

jobs are taken from the head of the queue. When a new job is released, it is placed in the queue according to its deadline.

EDD is an optimal algorithm for minimizing lateness in a single machine system. In our case, it corresponds to minimizing the number of rejected jobs. Gupta and Palin [12] showed that there cannot exist an algorithm with a competitive ratio greater than $1-(1/f_i)+\dot{o}$ with $m \geq 1$ machines, and $\dot{o} > 0$ is arbitrary small for the problem of allocating jobs on a hard real-time scheduling model in which a job must be completed if it was admitted for execution. They proposed an algorithm that achieves a competitive ratio of at least $1-(1/f_i)$ and demonstrated that this is an optimal scheduler for hard real-time scheduling with m machines. The admittance test also proposed by them consists in verifying that all the already accepted jobs whose deadline is greater than that of the incoming job will be completed before their deadline is met.

3.2 Workload

In order to evaluate the performance of SLA scheduling, we performed a series of experiments using traces of HPC jobs obtained from the Parallel Workloads Archive (PWA) [13] and the Grid Workloads Archive (GWA) [14].

These traces are logs from real parallel computer systems, and they give us a good insight in how our proposed schemes will perform with real users. Predominance of low parallel jobs in real logs is well known. Even though some jobs in the traces require multiple processors, we consider that in our model the machines have enough capacity to process them, so we can abstract their parallelism.

Since we assume that IaaS clouds are a promising alternative to computational centers, we can expect that workload submitted to clouds will have similar characteristics to the ones submitted to actual parallel and grid systems. In our log, we considered nine traces from DAS2 (University of Amsterdam), DAS2 (Delft University), DAS2 (Utrecht University), DAS2 (Leiden University), KTH, DAS2 (Vrije University), HPC2N, CTC, and LANL. Details of the log characteristics can be found in the PWA [13] and GWA [14].

To obtain valid statistical values, 30 experiments within one week period were simulated for each SLA. We calculated job deadlines based on the real processing time of the jobs.

4 Experimental Results

4.1 Single Machine Model

For the first set of experiments with a single machine system scheme, we performed experiments for 12 values of the slack factor: 1, 2, 5, 10, 15, 20, 25, 50, 100, 200, 500 and 1000. Although we do not expect that a real SLA provides slack factors greater than 50, large values are important to study expected system performance when slack factors tend to infinity.

Figures 1-5 show simulation results of SSL-SM algorithm. They present percentage of rejected jobs, total processing time of accepted jobs, mean waiting time, mean number of interruptions per job, and mean competitive factor**.**

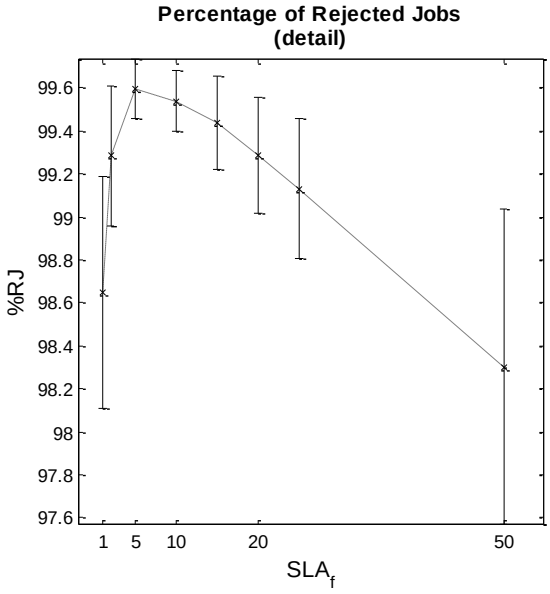

Fig. 1. Percentage of rejected jobs for SSL-SM algorithm

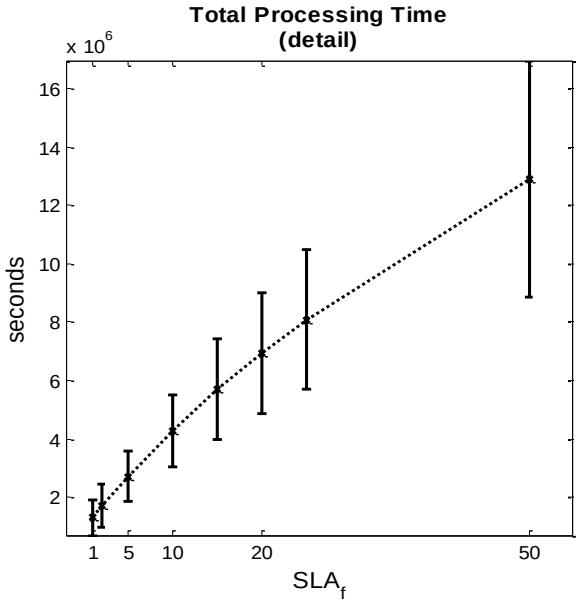

Fig. 2. Total processing time for SSL-SM algorithm

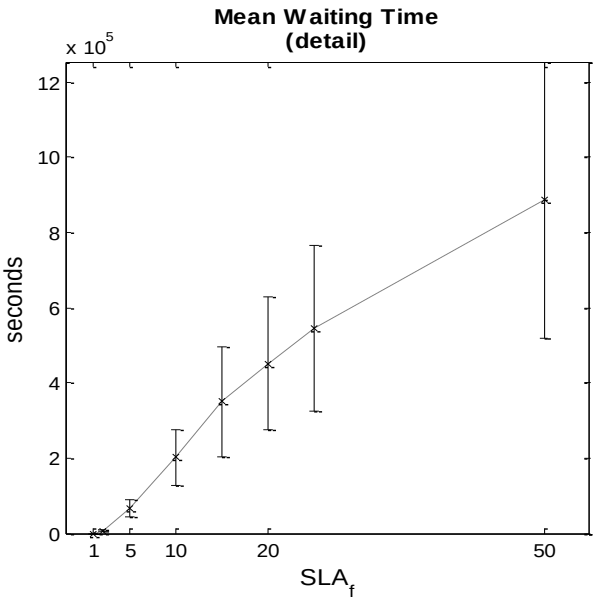

Fig. 3. Mean waiting time of jobs for SSL-SM algorithm

Figure 1 shows the percentage of rejected jobs for the SSL-SM algorithm. We see that the number of rejected jobs decreases while the slack factor increases.

Large values of slack factor increase the flexibility to accept new jobs by delaying the execution of already accepted ones. In the case when a slack factor is equal to 1, the system cannot accept new jobs until the job in execution is completed. We observe that the percentage of rejected jobs with a slack factor of 1 is a bit lower than that with values of slack factor from 2 to 25. However, it does not mean that this slack factor allows the system to execute more computational work as we see in Figure 2. Figure 2 shows the total processing time of accepted jobs for the given slack factors. We see that the processing time increases as the slack factor increases, meaning that the scheduler is able to exploit the increased flexibility of the jobs. Figure 3 shows mean waiting time versus the slack factor. It demonstrates that an increase of total processing time causes an increase of waiting time.

We also evaluate the mean number of interruptions per job; these results are showed in Figure 4. We see that for small slack factors the number of interruptions is greater than that for larger slack factors. Mean values are below 1 interruption per job. Moreover, if a slack factor is more than 10, the number of interruptions per job is stable and vary between 0.2 and 0.3. This fact is important; keeping the number of interruptions low prevents the system overhead.

Figure 5 shows the mean competitive factor. It represents the infrastructure provider objective to maximize his total income. Note that a larger competitive factor is better than a smaller one. When the slack factor is equal to 1, the competitive factor is 0.85. Once the slack factor is increased to 5, we obtain better competitive factors. When the slack factor is equal to 5, the mean competitive factor has its maximum value of 0.94. Passing this point, the competitive factor decreases when the slack factor is equal to 200. We consider that at this point the deadlines of the jobs are much larger than their processing time. If the slack factor is between 200 and 500, the competitive factor is increased again because the maximum deadline gets close to the sum of processing times.

When the deadline of all jobs tends to infinity, the completive factor is optimal as expected.

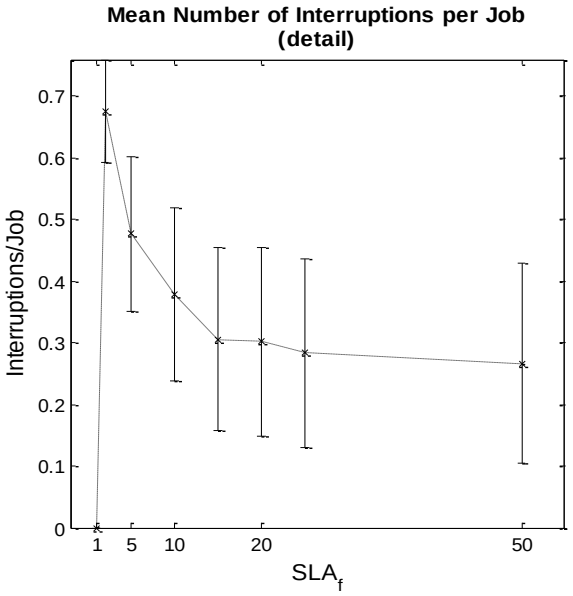

Fig. 4. Mean number of interruptions per job for SSL-SM algorithm

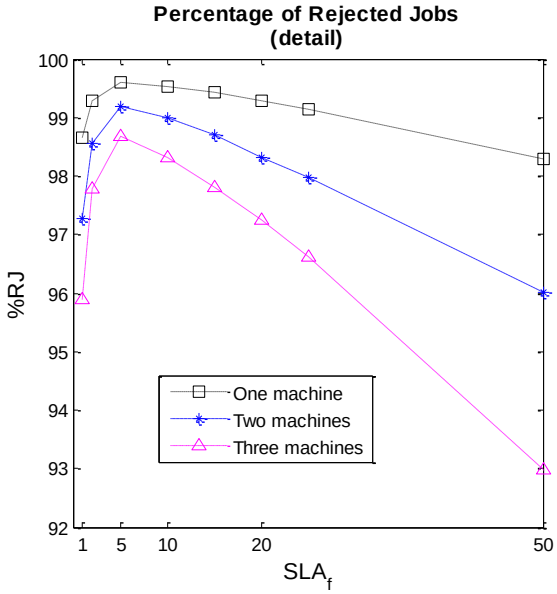

Fig. 6. Percentage of rejected jobs for SSL-PM algorithm

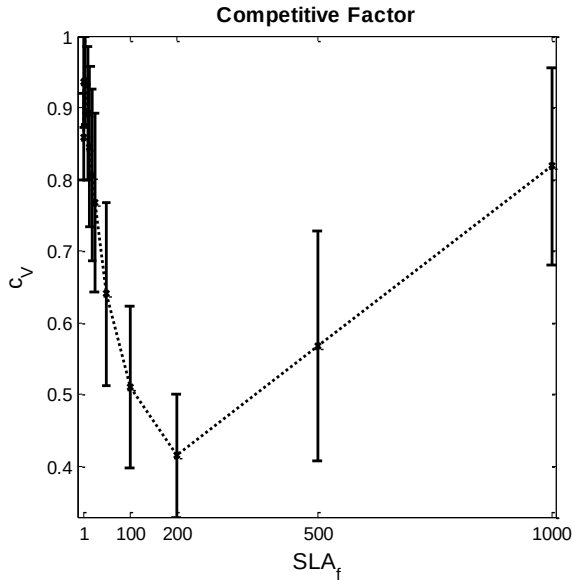

Fig. 5. Mean competitive factor of SSL-SM algorithm

In a real cloud scenario, the slack factor can be dynamically adjusted in response to changes in the configuration and/or the workload. To this end, historical workload within a given time interval can be analyzed to determine an appropriate slack factor. The time interval for this adjustment should be set according to the dynamic characteristics of the workload and in the IaaS configuration.

4.2 Multiple Machine Model

In this section, we present the results of SSL-PM algorithm simulations on two and three machines. We plotted the SSL-SM results to analyze the change of the system performance when the number of machines varies.

Figures 6-11 show the percentage of rejected jobs, total processing time of accepted jobs, mean waiting time, mean number of interruptions per job, efficiency and mean competitive factor**.**

Figure 6 presents the percentage of rejected jobs. It can be seen that an increase of the number of machines has a limited effect on the acceptability of jobs when the slack factor is small. However, larger values of slack factor have greater impact on the number of accepted jobs. Figure 7 shows the total processing time of accepted jobs. The processing time is increased

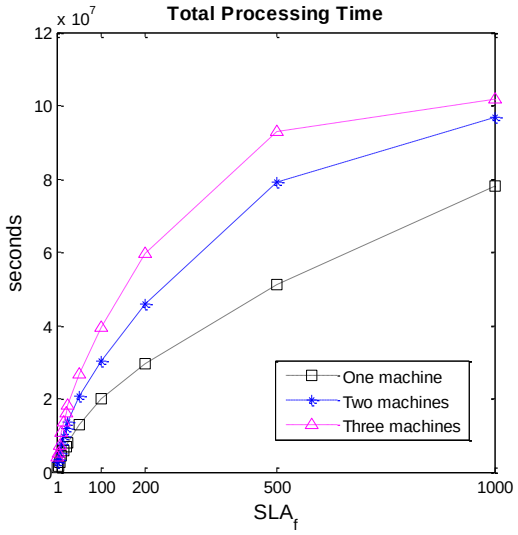

Fig. 7. Total processing time for SSL-PM algorithm

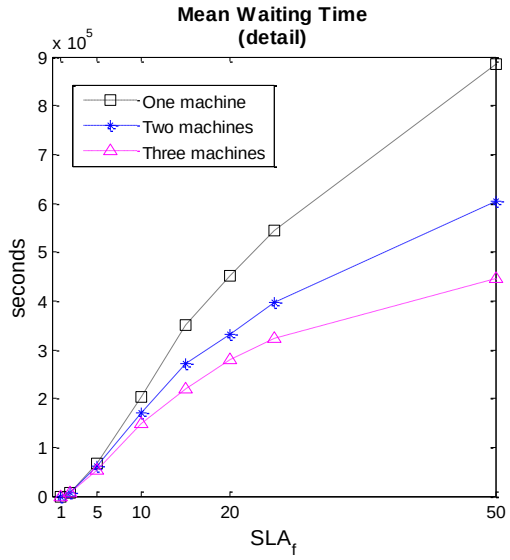

Fig. 8. Mean waiting time for SSL-PM algorithm

as more machines are added to the system. However, doubling and tripling the processing capacity do not cause the same increase in the processing time. This effect can be clearly seen when the slack factor is large. We conclude that an increase in the processing capacity will be more effective with smaller slack factors. Figure 8 shows the mean waiting time when slack factor

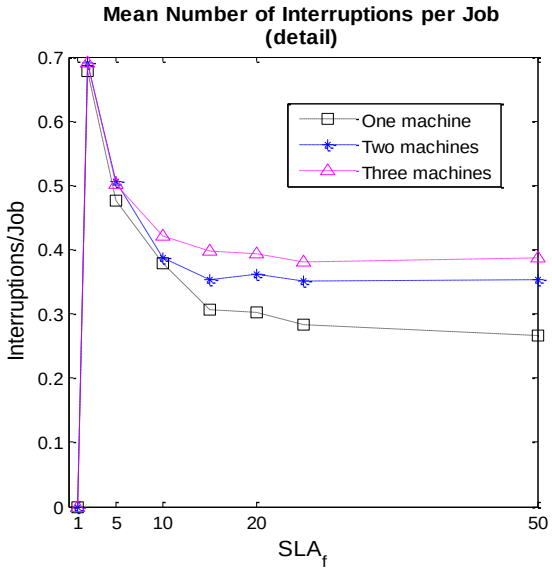

Fig. 9. Mean number of interruption for SSL-PM algorithm

varies. We see that an increase of the total processing time, as a result of larger slack factors, also causes an increase of waiting time. Additionally, adding more machines to the system makes the increase of the mean waiting time less significant.

Figure 9 shows the mean number of interruptions per job. We see that an increase of the number of machines increases the number of interruptions. This increase is not considerable, and is stabilized as the slack factor is increased. The number of interruptions is maximal with a slack factor of 2 for all three models. Figure 10 shows the execution efficiency. This metric indicates the relative amount of useful work which the system executes during the interval between the release time of the first job and the completion of the last job.

We see that a decrease of efficiency, at least with moderate slack factors, mainly depends on the number of machines. Figure 11 presents the competitive factor while the slack factor varies. We see that for the two and three machine system configuration the maximum competitive factor is obtained with a slack factor of 2. As we already mentioned, in the case of a single machine configuration the best competitive

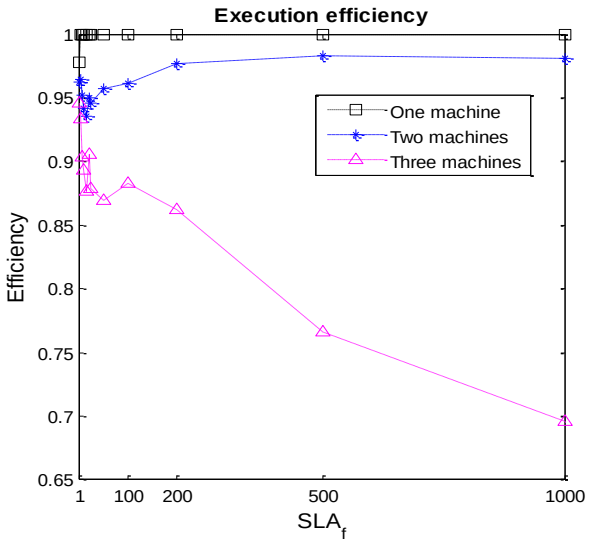

Fig. 10. Execution efficiency for SSL-PM algorithm

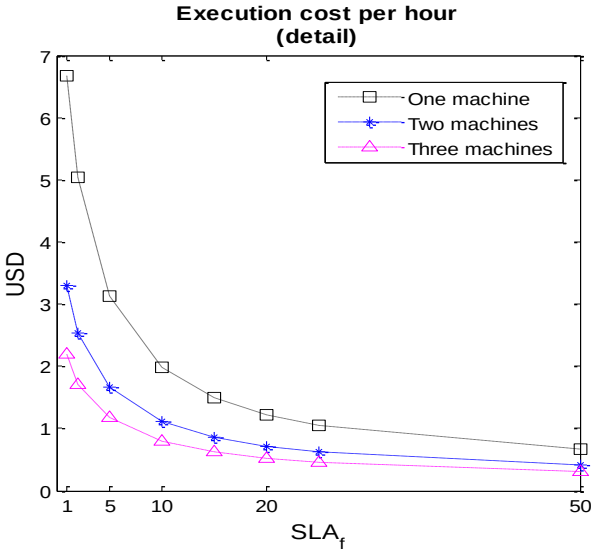

Fig. 12. Execution cost per hour

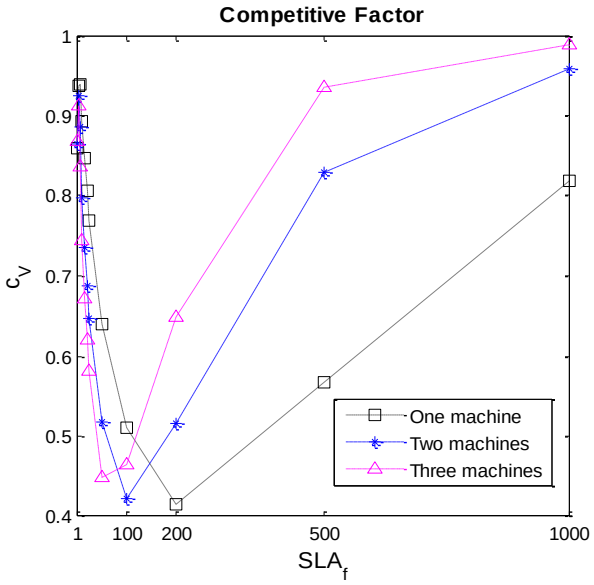

Fig. 11. Competitive factor for SSL-PM algorithm

factors are obtained with a slack factor of 2 and 5. We can also observe that when the slack factor is increased, the competitive factor is decreased. This happens until the slack factor becomes large enough to create a significant difference between job deadlines and their processing times. This is clearly seen when the slack factor is 200 for a single machine configuration, and 100 for two and three machines.

In the cases of two and three machines configuration, for the slack factor greater than 500, the competitive factor almost reached the optimal value.

4.3 Execution Costs

In the IaaS scenario, cloud providers offer computer resources to customers on a pay-as-you-go basis. The price per time unit depends on the services selected by the customer. This charge depends not only on the price the user is willing to accept, but also on the cost of the infrastructure maintenance.

In order to estimate this charge, we propose a tariff function that depends on the slack factor. We first take into account that the provider needs to recover the maintenance cost from the execution of jobs. We assume that the provider pays a flat rate for the use/maintenance of the resources.

The total maintenance cost of job processing (co_t) can be calculated using the expression

$\frac{\sum_{j=1}^{n_r} p_j}{m} \times u_u \times m$. The cost per time unit co_u can be calculated as $co_u = \frac{co_t}{\sum_{j=1}^{n} p_j}$, where $\sum_{j=1}^{n_r} p_j$ is the sum of processing times of all released jobs, u_u is the price per unit of maintenance, m is the number of machines, and $\sum_{j=1}^{n} p_j$ is the sum of processing times reached by the algorithm. We consider that u_u is equal to 8.5 cents per hour, which is the price that Amazon EC2 charges for a small processing unit [15].

Figure 12 shows the execution cost per hour when the slack factor varies. As it can be seen, the cost of processing jobs with a small slack factor is larger than the execution of jobs with a looser slack factor. Moreover, the costs are larger if fewer machines are used. The reason is that a system with less machines and a small slack factor rejects most of the jobs within a given interval, so the execution is costly. Therefore, configurations that execute more jobs have lower costs per execution time unit. Clear profit is generated if cost per time unit is incremented.

5 Conclusions and Future Work

The use of Service Level Agreements (SLAs) is a fundamentally new approach for job scheduling. According to this approach, scheduling is based on satisfaction of QoS constraints. The main idea is to provide different levels of service, each addressing a different set of customers. While a large number of service levels leads to high flexibility for customers, it also produces a significant management overhead. Hence, a suitable tradeoff must be found and adjusted dynamically, if necessary. While theoretical worst case IaaS scheduling models begin to emerge, fast statistical techniques applied to real data are effective as have been shown empirically.

In this paper, we presented an experimental study of two greedy acceptance algorithms, namely, SSL-SM and SSL-PM, with known worst case performance bounds. They are based on the adaptation of the preemptive EDD algorithm for job scheduling with different service levels on different number of machines.

Our study results in several contributions. Firstly, we identified several service levels to make scheduling decisions with respect to job acceptance; secondly, we considered and analyzed two test cases on a single machine and on parallel machines; thirdly, we estimated the cost function for different service levels; then, we showed that the slack factor can be dynamically adjusted in response to changes in the configuration and/or the workload. To this end, the past workload within a given time interval can be analyzed to determine an appropriate slack factor. The time interval for this adaptation depends on the dynamics of the workload characteristics and IaaS configuration.

Though our model of IaaS is simplified, it is still a valid basic abstraction of SLAs that can be formalized and treated automatically.

In this paper, we explored only a few scenarios of using SLAs. The IaaS clouds are usually large scale and vary significantly. It is not possible to satisfy all QoS constraints from the service provider perspective if a single service level is used. Hence, a balance between the number of service levels and the number of resources needs to be found and adjusted dynamically. A system can have several specific service levels (e.g., Bronze, Silver, Gold) and algorithms to keep the system with QoS specified in SLA. However, further study of algorithms for multiple service classes and the resource allocation algorithms is required to assess their actual efficiency and effectiveness. This will be the subject of future work to achieve a better understanding of service levels in IaaS clouds. Moreover, other scenarios of the problem with different types of SLAs and workloads with a combination of jobs with and without SLA still need to be addressed. Also, as future work, we will consider the elasticity of slack factors in order to increase profit while providing better QoS to users.

References

1. **Garg, S.K., Gopalaiyengar, S.K., & Buyya, R. (2011).** SLA-based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter. *11th international conference on Algorithms and Architectures for parallel processing* (ICA3PP'11), Melbourne, Australia, 371–384.
2. **Wu, L., Garg, S.K., & Buyya, R. (2011).** SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments. *11th IEEE/ACM International* Symposium on *Cluster, Cloud and Grid Computing (CCGrid 2011),* Newport Beach, CA., USA, 195–204.
3. **Patel, P., Ranabahu, A., & Sheth, A. (2009).** Service Level Agreement in Cloud Computing (Technical Report). Ohio Center of Excellence in Knowledge-enabled Computing.
4. **Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., & Xu, M. (2004).** Web services agreement specification (WS-Agreement), (GFD-R-P.107). *Global Grid.*
5. Review and summary of cloud service level agreements. (s.f.). Retrieved from http://www.ibm.com/developerworks/cloud/library/cl-rev2sla.html.
6. **Wu, L., Garg, S.K., & Buyya, R. (2011).** SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments. *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011),* Newport Beach, CA,USA, 195–204.
7. **Freitas, A.L., Parlavantzas, N., & Pazat, J.L. (2011).** Cost Reduction Through SLA-driven Self-Management. *Ninth IEEE European Conference on Web Services (ECOWS),* Lugano, Switzerland, 117–124.
8. **Silaghi, G.C., Şerban, L.D., & Litan, C.M. (2010).** A Framework for Building Intelligent SLA Negotiation Strategies under Time Constraints. *Economics of Grids, Clouds, Systems, and Services*, *Lecture Notes in Computer Science,* 6296, 48–61.
9. **Macías, M., Smith, G., Rana, O., Guitart, J., & Torres, J. (2010).** Enforcing Service Level Agreements Using an Economically Enhanced Resource Manager. *Economic Models and Algorithms for Distributed Systems, Autonomic Systems,* 109–127.
10. **Baruah, S.K. & Haritsa, J.R. (1997).** Scheduling for overload in real-time systems. *IEEE Transactions on Computers*, 46(9), 1034–1039.
11. **Schwiegelshohn, U. & Tchernykh, A. (2012).** Online Scheduling for Cloud Computing and Different Service Levels. *IEEE 26th International Parallel and Distributed Processing Symposium Workshops,* Shanghai, China, 1067–1074.
12. **Gupta, B.D. & Palis, M.A. (2001).** Online real-time preemptive scheduling of jobs with deadlines on multiple machines. *Journal of Scheduling,* 4(6), 297–312.
13. **D. Feitelson (2008).** Parallel Workloads Archive. *Algorithms and Architectures for*.
14. **Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., & Epema, D.H. (2008).** The Grid Workloads Archive. *Future Generation Computer Systems,* 24(7), 672–686.
15. **Amazon Services. (2013).** Precios deAmazon EC2 *Retrieved from http://aws.amazon.com/ec2/pricing/*.

Anuar Lezama Barquet obtained a degree in Electric and Electronic Engineer from the National Autonomous University of Mexico (UNAM). He received his M.S. in Computer Science from the CICESE Research Center in 2012. His interests include parallel computing, scheduling, and cloud computing.

Andrei Tchernykh is a researcher at the Computer Science Department, CICESE Research Center, Ensenada, Baja California, Mexico. From 1975 to 1990 he was with the Institute of Precise Mechanics and Computer Technology of the Russian Academy of Sciences (Moscow, Russia). He received his Ph.D. in Computer Science in 1986. In CICESE, he is a coordinator of the Parallel Computing Laboratory. He is a member of the National System of Researchers of Mexico

(SNI), Level II. He leads a number of national and international research projects. He served as a program committee member of several professional conferences and a general co-chair for international conferences on Parallel Computing Systems. His main research interests include scheduling, load balancing, adaptive resource allocation, scalable energy-aware algorithms, green grid and cloud computing, eco-friendly P2P scheduling, multi-objective optimization, scheduling in real time systems, computational intelligence, heuristics, meta-heuristics, and incomplete information processing.

Ramin Yahyapour is executive director of the GWDG University of Göttingen. He has done research in Clouds, Grid and Service-oriented Infrastructures for several years. His research interests are in resource management. He is a steering group member and on the Board of Directors in the Open Grid Forum. He has participated in several national and European research projects. Also, he is a scientific coordinator of the FP7 IP SLA@SOI and was a steering group member in the CoreGRID Network of Excellence.

Article received on 22/02/2013; accepted 01/08/2013.