Scheduling Different Types of Applications in a SaaS Cloud

Georgios L. Stavrinides and Helen D. Karatza
Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
{gstavrin, karatza}@csd.auth.gr

Keywords: Scheduling, Complex Workload, SaaS Cloud, Performance, Modeling, Simulation.

Abstract: As Software as a Service (SaaS) cloud computing gains momentum, the efficient scheduling of different types of applications in such platforms is of great importance, in order to achieve good performance. In SaaS clouds the workload is usually complex and comprises applications with various degrees of parallelism and priority. Therefore, one of the major challenges is to cope with the case where high-priority real-time single-task applications arrive and have to interrupt other non-real-time parallel applications in order to meet their deadlines. In this case, it is required to effectively deal with the real-time applications, at the smallest resulting degradation of parallel performance. In this paper, we investigate by simulation the performance of strategies for the scheduling of complex workloads in a SaaS cloud. The examined workload consists of non-real-time applications featuring fine-grained parallelism (gangs) and periodic high-priority soft real-time single-task applications that can tolerate deadline misses by bounded amounts. We examine the impact of gang service time variability on the performance of the scheduling algorithms, by considering service demands that follow a hyper-exponential distribution. The simulation results reveal that the relative performance of the employed scheduling strategies depends on the type of the workload.

1 INTRODUCTION

Software as a Service (SaaS) cloud computing has become prevalent in recent years, replacing the traditional software delivery model, which required the installation of the software on the client's servers. According to this emerging software distribution approach, which may be considered as the evolution of the Service-Oriented Architecture (SOA) model, the software is hosted by the vendor and made available to the end-users over the Internet, as a cloud service (Cusumano, 2010) and (Beloglazov et al., 2012).

Some of the main benefits of the SaaS cloud computing model are:

- The software is always up to date.
- The software can be accessed via various devices and from different locations.
- It provides overall easier administration and maintenance, as the SaaS clients do not have to acquire, maintain and monitor expensive hardware and software infrastructure.

The clients use the software services, without any control on the host environment, either on a pay-as-you-go basis or a subscription based pricing model (Bittencourt et al., 2012) and (Dillon et al., 2010).

1.1 Motivation

SaaS cloud computing usually relies on a *multi-tenant* model, where applications of different users run on the same virtual machines (VMs). Due to the tremendous increase of users and the number of different applications sharing the underlying virtualized resources, the performance of SaaS clouds has become a crucial area of research. Therefore, the efficient scheduling of different types of applications in such platforms is of great importance, in order to effectively utilize the underlying multi-tenant infrastructure and achieve good performance, while maintaining a certain level of *Quality of Service (QoS)* (Hofer and Karagiannis, 2011), (Rimal et al., 2009) and (Stavrinides and Karatza, 2015).

In SaaS clouds the workload is usually complex and comprises applications with various degrees of parallelism and priority. Consequently, one of the major challenges is to cope with the case where high-priority real-time single-task applications arrive and have to interrupt other non-real-time parallel applications in order to meet their deadlines. In this case, it is required to effectively deal with the real-time applications, at the smallest resulting degradation of parallel performance. Moreover, part of the workload may consist of fine-grained parallel

Stavrinides G. and Karatza E.
Scheduling Different Types of Applications in a SaaS Cloud.
DOI: 10.5220/0006223101440151
In *Proceedings of the Sixth International Symposium on Business Modeling and Software Design (BMSD 2016)*, pages 144-151
ISBN: 978-989-758-190-8

applications that present high variability in their service times. In this case, the scheduling algorithm should cope with bursts of very small service times and a few but very large ones, compared to the average service time of applications.

1.2 Contribution

In this paper, we focus on the performance of strategies for the scheduling of complex workloads in a SaaS cloud with multi-tenant VMs. The examined workload consists of non-real-time applications featuring fine-grained parallelism and periodic high-priority soft real-time single-task applications that can tolerate deadline misses by bounded amounts.

We examine the impact of service time variability of parallel applications on the performance of the scheduling algorithms, by considering service demands that follow a hyper-exponential distribution. To our knowledge, scheduling complex workloads, including fine-grained parallel applications and periodic soft real-time single-task jobs, in such a framework, does not appear elsewhere in the research literature.

1.3 Background and Related Work

One type of workload submitted to SaaS clouds is *bag-of-tasks (BoT)* applications. Each BoT is a collection of independent tasks that do not communicate with each other and can run in any order (Karatza, 2004), (Kim et al., 2007), (Moschakis and Karatza, 2015), (Papazachos and Karatza, 2015) and (Terzopoulos and Karatza, 2016). Another type of workload is applications that consist of interdependent tasks, which often have precedence constraints among them and thus form a *directed acyclic graph (DAG)* (Stavrinides and Karatza, 2010), (Stavrinides and Karatza, 2011), (Stavrinides and Karatza, 2012) and (Stavrinides and Karatza, 2014).

However, most often the workload in SaaS clouds consists of fine-grained parallel applications. They consist of frequently communicating tasks, which are scheduled to run simultaneously on different VMs, as *gangs*. Gang scheduling is an efficient resource management technique in the case where parallel applications feature fine-grained parallelism.

Since with gang scheduling a gang's task can start execution only if all of the other sibling tasks can also start processing, some VMs may remain idle even when there are tasks waiting in their queues. This problem becomes more complex when the workload also includes real-time applications which have higher priority than gangs.

Gang scheduling algorithms have been proposed and studied by many authors, each differing in the way resources are shared among the jobs (Stavrinides and Karatza, 2008), (Stavrinides and Karatza, 2009), (Streit, 2005) and (Zhang et al., 2003).

In (Karatza, 2006), the performance of two well-known gang scheduling methods is studied, the *Adapted-First-Come-First-Served (AFCFS)* and the *Largest-Job-First-Served (LJFS)*. It has been shown that in many cases LJFS performs better than AFCFS. However, in this case no real-time jobs are considered and the overall performance is expressed by the average response time of gangs.

In this paper, we study gang scheduling in a queueing network model of a SaaS cloud. The performance of two policies that are variations of AFCFS and LJFS, is studied under various work-loads, which include gangs, as well as periodic real-time single-task jobs that have higher priority than gangs. The periodic single-task jobs are considered as soft-real time applications, in the sense that they can tolerate deadline misses by bounded amounts.

Related research includes (Karatza, 2007) and (Karatza, 2008). These papers examine critical sporadic jobs that need to start execution upon arrival and therefore they interrupt gangs. In this paper, we examine periodic soft real-time jobs that can tolerate some delay called *slack time*. This happens in the case when gangs in service require only a small amount of time to finish their execution that is less than or equal to the slack time.

Scheduling workloads consisting of soft-real time jobs and gangs in distributed systems has also been studied in (Karatza, 2014). However, the workload considered in this case does not include parallel applications with highly variable service demands.

Tardiness bounds for sporadic real-time task systems have been studied in (Devi and Anderson, 2006) and (Leontyev and Anderson, 2010). However, these papers do not consider gangs in their workloads.

In this paper, we examine the impact of gang service time variability on the performance of the scheduling algorithms, by considering service demands that follow a hyper-exponential distribution. A high variability in task service demand implies that there is a proportionately large number of service demands that are very small compared to the mean service time and a comparatively small number of service demands that

are very large.

When a gang with a long service demand starts execution, it occupies its assigned VMs for a long time interval, and depending on the scheduling policy that is employed, it may introduce inordinate queueing delays for other tasks waiting for service.

The performance evaluation of complex distributed systems such as clouds is often possible only by simulation rather than by analytical techniques, due to the complexity of the systems. Simulation can provide important insights into the efficiency and tradeoffs of scheduling in such environments. Therefore, due to the complexity of the system under study, we use discrete event simulation to evaluate the performance of the scheduling algorithms.

The remainder of this paper is organized as follows: Section 2 introduces the model and the methodology employed, describing the system and workload models. Sections 3 and 4 describe the routing and gang scheduling policies respectively, whereas Section 5 presents the performance metrics. The model implementation and its input parameters, as well as the simulation results, are presented and analysed in Section 6. Finally, Section 7 summarizes the paper and provides future research directions.

2 MODEL AND METHODOLOGY

2.1 System and Workload Models

This paper uses a simulation model to address performance issues. The target SaaS cloud is considered to consist of a set V of p virtual machines $V = \{VM_1, VM_2, \ldots, VM_p\}$, that are fully connected by a virtual network (Figure 1).

It is assumed that the communication between the virtual machines is contention-free. Each virtual machine VM_i serves its own queue of tasks and has mean execution rate μ. The VMs in the cloud are multi-tenant, that is each VM processes tasks of applications submitted by different users.

Each parallel job is submitted for execution in the cloud by a user. Users share the virtual machines to run concurrently their applications. Since the parallel applications are submitted dynamically by multiple users, it is assumed that they arrive in the cloud in a Poisson stream with mean arrival rate λ.

The number of tasks in a parallel job x is the job's *degree of parallelism* and it is represented as $t(x)$. In Figure 1 virtual machines are allocated to a parallel job x, which has j parallel tasks. VM_p is assigned to a periodic single-task job.

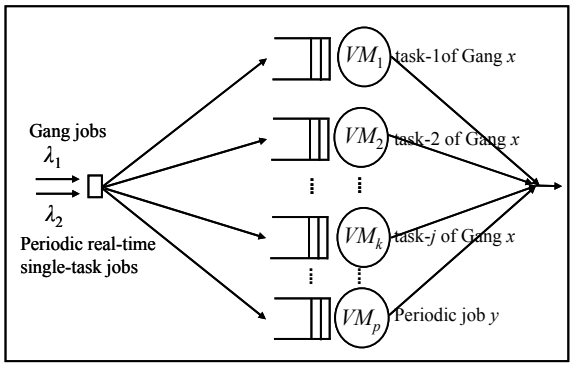

Figure 1: The queueing network model.

If $vm(x)$ represents the number of VMs required by job x, then the following relationship holds: $1 \leq t(x) = vm(x) \leq p$. We call *size* of job x the number of its parallel tasks. A job is *small* (*large*) if it consists of a small (large) number of tasks. Soft real-time jobs are periodic single-task jobs.

Each task of a gang x is routed to a different VM. Gang x starts to execute only if all of the $vm(x)$ VMs assigned to its tasks are available. Otherwise, all of the x gang tasks wait in their assigned queues. When a gang terminates execution, all VMs assigned to it are released.

An important issue that arises is the need to serve a real-time single-task job on a virtual machine executing a gang task. Therefore, a mechanism is needed to deal with the fact that resources allocated to a gang are changing. The scheduler must give priority to real-time applications, but at the same time provide good performance for parallel applications that compete for resources.

For each soft real-time application there is a slack time during which it can wait for a gang that needs a small amount of time to finish execution. The slack time is a constant tardiness that is independent of elapsed time. If the time until the completion of a gang is larger than the slack time, then the job scheduler interrupts the gang upon the arrival of a soft real-time application, so that it can occupy a VM. The remaining virtual machines that are assigned to the interrupted gang can serve tasks of other gangs that are waiting at their queues. All of the work that was performed on all tasks associated with the interrupted gang must be redone. The tasks of an interrupted parallel application are rescheduled for execution at the head of their assigned queues.

A technique used to evaluate the performance of the scheduling disciplines, is experimentation using a synthetic workload. The workload considered here is characterized by the following parameters:

- The distribution of the number of tasks of gangs.

- The distribution of gang tasks service demand.
- The distribution of soft real-time jobs service demand.
- The mean inter-arrival time of gangs.
- The period of soft real- time jobs.

2.2 The Distribution of Gang Sizes

We assume that the number of gang tasks is uniformly distributed in the range [1, p]. Therefore, the mean number of tasks per parallel application is equal to $m = (1+p)/2$.

2.3 Service Time Distribution

For the service demands of parallel applications and real-time jobs we assume that:

- Service demands of gang tasks follow a hyper-exponential distribution with a coefficient of variation CV and a mean $1/\mu$. CV determines the degree of variability. This is the ratio of the standard deviation of task execution time to its mean. CV >1 implies a large variability, compared to the variability of the exponential distribution.
- Real-time jobs service demands are exponentially distributed with a mean of $1/\mu$.

2.4 Distribution of Job Inter-arrival Times

We consider two arrival streams, one for gangs and one for real-time jobs:

- The inter-arrival times of gangs are exponential random variables with a mean of $1/\lambda_1$.
- The inter-arrival times of soft real-time jobs are constant time intervals equal to $1/\lambda_2$ (i.e. the real-time jobs are periodic).

3 ROUTING POLICIES

Job routing in SaaS clouds is an important aspect for good performance. In this paper, our workload model consists of two different types of jobs. Therefore, we examine two different routing policies, one for each job type.

The first job type concerns the parallel applications, which are gangs. Hence, in this case routing occurs at the task level. The second job type concerns the single task real-time applications. Consequently, in this case routing occurs at the job level.

3.1 Parallel Applications Routing

A variation of the *join the shortest queue* policy is used. That is, the $t(x)$ tasks of a gang x are assigned to the shortest $t(x)$ of the p queues, every task to a different VM queue.

3.2 Real-time Applications Routing

The routing policy of real-time jobs is *join the shortest queue*. In the experiments that we have conducted for this research, appropriate values of the period have been chosen, so that there are no further real-time job arrivals in the case where all of the VMs serve real-time jobs.

4 PARALLEL APPLICATIONS SCHEDULING POLICIES

It is assumed that the scheduler knows the exact number of VMs required by each gang. The following two scheduling strategies are employed in our simulations.

4.1 Adapted-First-Come-First-Served with Execution Interruption based on Slack Time (AFCFS-ST)

This strategy schedules a gang job whenever VMs assigned to its tasks are available. When there are not enough VMs available for a large job whose tasks are waiting in the front of the queues, then the AFCFS-ST policy schedules tasks of smaller jobs in the queues. A problem with this scheduling policy is that it tends to favor small gangs at the expense of larger gangs and thus it may increase system fragmentation. A gang job may be interrupted upon arrival of a real-time single-task job.

4.2 Largest-Job-First-Served with Execution Interruption based on Slack Time (LJFS-ST)

With this strategy tasks that belong to larger gangs are placed at the head of queues. All gang tasks in queues are searched in order, and the first jobs whose assigned VMs are available begin execution. This scheduling method favors large, highly parallel gangs at the expense of smaller gangs, but in many cases this treatment of large gangs is acceptable. For example, supercomputers are often used to particularly run highly parallel jobs for fast

execution. However, LJFS-ST involves an overhead because the VM queues need to be re-arranged each time a new gang is added. Similarly to the AFCFS-ST case, a gang job may be interrupted upon arrival of a real-time job.

In both cases of AFCFS-ST and LJFS-ST, when a real-time job terminates, the interrupted gang may not resume execution immediately, as some of the VMs assigned to its tasks may be running other jobs. Those jobs will not terminate at the same time so the interrupted gang will not use the assigned VMs efficiently. It is worth noting that when a real-time job is assigned to a virtual machine, it is not only the gang tasks of the particular VM queue that are delayed, but also gang tasks in other queues that have a sibling task waiting in the particular queue.

5 PERFORMANCE METRICS

The response time rt_i of a gang i is the time interval from the dispatching of its tasks to different VM queues, to the service completion of the gang − this is the time spent in VM queues plus the time spent in service.

The average response time RT of n gangs is defined as:

$$RT = \frac{\sum_{i=1}^{n} rt_i}{n} \quad (1)$$

Additionally, we weight each gang's response time with its size (Streit, 2005). Consequently, it is avoided that gangs with the same execution time, but with different number of tasks, have the same impact on performance. The average weighted response time WRT of n gangs is defined as:

$$WRT = \frac{\sum_{i=1}^{n} vm(x_i) \times rt_i}{\sum_{i=1}^{n} vm(x_i)} \quad (2)$$

The parameters used in the simulation experiments are shown in Table 1.

Table 1: System parameters and performance metrics.

ST	Slack time of real-time jobs
p	Number of VMs
m	Average number of tasks per parallel job
μ	Mean execution rate of a VM
$1/\mu$	Mean execution time of a VM
CV	Coefficient of variation of service demands
λ_1	Mean arrival rate of gangs
$1/\lambda_2$	Period of real-time jobs
U	Average VM utilization
RT	Average response time of gangs
D_{RT}	Relative (%) decrease in RT when LJFS-ST method is employed instead of the AFCFS-ST policy
WRT	Average weighted response time of gangs
D_{WRT}	Relative (%) decrease in WRT when LJFS-ST method is employed instead of the AFCFS-ST policy

6 SIMULATION RESULTS AND DISCUSSION

6.1 Model Implementation and Input Parameters

The queueing network model described in Section 2 is implemented via discrete event simulation. Due to the complexity of the system and the workload model under study, we implemented our own discrete event simulation program in C, tailored to the requirements of the specific case study.

In order to derive the mean values of the performance parameters we used the independent replications method. For each set of workload parameters we run 30 replications of the simulation with different seeds of random numbers and for 32,000 served jobs in each replication. We considered this simulation length long enough to derive results, as we found by experimentation that longer runs did not affect simulation output significantly. The use of sufficiently long simulation runs, is one of the ways to reduce the effect of initial bias on simulation results.

For every mean value of the performance parameters, a 95% confidence interval was evaluated. The half-widths of all confidence intervals are less than 5% of their respective mean values evaluated. For our experiments, we used the simulation input parameters shown in Table 2.

In our workload model there are on average $m = (p+1)/2 = 8.5$ tasks per parallel job. Therefore, if we do not consider any real-time jobs and all VMs are busy serving gangs, then an average of $p / m = 1.88235$ parallel jobs can be served per each unit of time. This implies that we should choose a λ_1 such that the condition $\lambda_1 < 1.88235$ holds, so that the VM queues will not be saturated. However, due to the real-time job arrivals, the number of VMs that are

available for gang service is $[p - (\lambda_2/\mu)]$. Therefore, we have to choose a value of λ_1 for which the following relationship holds:

$$\lambda_1 < [p - (\lambda_2/\mu)] / m \quad (3)$$

However, due to gang scheduling, there are often idle VMs, despite the fact that there are gang tasks waiting in the respective queues. Therefore, the VM queues get very easily saturated when the mean inter-arrival time of gangs is close to $[p - (\lambda_2/\mu)] / m$. After experimental runs with various values of $1/\lambda_1$, we chose 0.78 as the smallest mean inter-arrival time of gangs for the experiments, for all cases of $1/\lambda_2$. The input parameter values which were used for the experiments are presented in Table 2.

Table 2: Input parameters.

ST	0.1
$1/\mu$	1
$1/\lambda_1$	0.84, 0.78
$1/\lambda_2$	20, 30, 40
CV	2

6.2 Performance Analysis

The simulation results that are presented next describe the relative performance of the two gang scheduling policies.

6.2.1 Performance with Regard to Parallel Applications Service

Figure 2 presents D_{RT} versus $1/\lambda_1$ in the cases of period $1/\lambda_2$ = 20, 30 and 40. Figure 3 presents D_{WRT} versus $1/\lambda_1$ in the cases of period $1/\lambda_2$ = 20, 30 and 40.

Figures 2 and 3 show that for both arrival rates of gangs and for all cases of real-time jobs period, the LJFS-ST method yields lower mean response time than AFCFS-ST.

In each real-time job period case, the superiority of LJFS-ST over AFCFS-ST is more significant in the case of high load ($1/\lambda_1$ = 0.78). This is due to the fact that the advantages of the LJFS-ST case are better exploited when there is a sufficient number of gangs in the queues, so that they can be selected according to the LJFS-ST criteria.

The simulation results also reveal that the D_{RT} increase with increasing load is more significant in the larger periods. This may be explained by the fact that the smaller the period of real-time jobs, the larger is the possibility that fewer virtual machines are available for gang service. Therefore, the potential of the LJFS-ST policy is not completely exploited, as large applications cannot find enough VMs available to serve them.

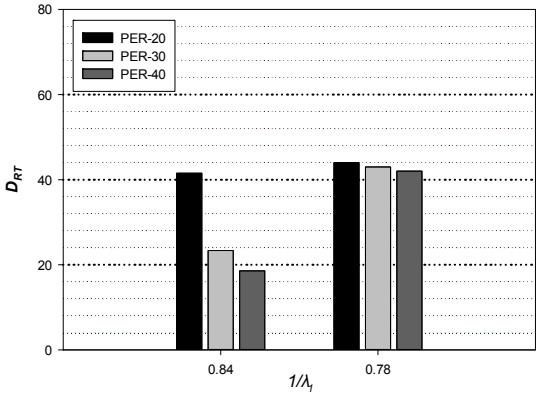

Figure 2: D_{RT} ratio versus $1/\lambda_1$, soft real-time jobs with period 20, 30 and 40.

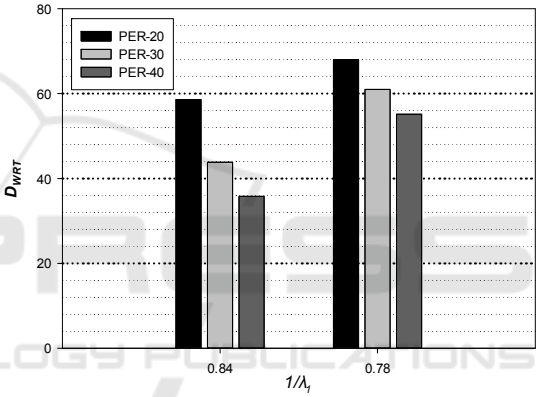

Figure 3: D_{WRT} ratio versus $1/\lambda_1$, soft real-time jobs with period 20, 30 and 40.

In each case of the arrival rate of gangs, D_{RT} decreases with increasing period size. This is because when real-time jobs arrive less frequently, then a smaller number of gangs are interrupted. The D_{RT} decrease with increasing period size is smaller in the case of heavier load ($1/\lambda_1$ = 0.78). This is due to the fact that when a real-time job blocks a gang, then it does not only affect the performance of the blocked gang, but it also affects the performance of a number of subsequent gangs.

Figure 3 shows that the observations that hold for the relative performance of the scheduling policies in terms of the average response time of parallel applications, also generally hold in terms of the average weighted response time. In all cases D_{WRT} is larger than D_{RT}. Therefore, the superiority of LJFS-ST over AFCFS-ST appears more significant when the job response time is weighted by the degree of parallelism of the jobs.

6.2.2 Performance with Regard to VM Utilization

Tables 3 and 4 depict the mean virtual machine utilization U versus $1/\lambda_1$, for period size $1/\lambda_2 = 20$, 30 and 40 respectively. In most cases, the mean VM utilization is either equal for both of the scheduling policies or it is very slightly larger in the LJFS-ST case. This is because the LJFS-ST policy schedules large gangs first and consequently, it schedules jobs on the available VMs more efficiently than the AFCFS-ST policy.

Table 3: U for AFCFS-ST.

$1/\lambda_1$	0.84	0.78
PER-20	0.711	0.740
PER-30	0.672	0.716
PER-40	0.636	0.690

Table 4: U for LJFS-ST.

$1/\lambda_1$	0.84	0.78
PER-20	0.712	0.768
PER-30	0.672	0.718
PER-40	0.637	0.695

However, with both scheduling strategies, part of the virtual machines utilization is comprised of repeated gang work due to the real-time job service. The amount of the repeated work depends on the number of gang tasks, the service demand of each task and the work that has already be done by the gang at the moment of interruption. It is possible for a gang to be interrupted several times during its execution. This would be caused by multiple real-time job arrivals at different virtual machines serving tasks of the same gang.

For each λ_1, the utilization slightly decreases with increasing period size. This is due to the fact that it is more possible to have to restart the parallel applications execution when the period of real-time applications is small, than when it is large. However, due to the variability in gang service demand in our workload model, only the gangs that have very large service demands may experience multiple service interruptions.

7 CONCLUSIONS AND FURTHER RESEARCH

This paper examines the performance of two gang scheduling policies, the LJFS-ST and the AFCFS-ST, in a SaaS cloud in the presence of periodic soft real-time single-task jobs. The objective is to enhance the performance of parallel applications, assuming that soft real-time jobs can tolerate a delay.

Simulation results show that the relative performance of the two scheduling methods depends on the workload. With regard to the performance metrics considered in this paper, the LJFS-ST method performs better than the AFCFS-ST, in all cases of workloads, providing promising results.

Furthermore, the superiority of LJFS-ST over AFCFS-ST appears more significant when the response time of parallel applications is weighted by their degree of parallelism.

Our future research plans include the examination of cases where along with gangs there are also workflow applications. Moreover, we plan to consider additional distributions for service demands and investigate their impact on the monetary cost charged to the end-users of the SaaS cloud.

REFERENCES

Cusumano, M., 2010. Cloud computing and SaaS as new computing platforms. *Communications of the ACM*. ACM, 53(4), 27-29.

Beloglazov, A., Abawajy, J. and Buyya, R., 2012. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*. Elsevier, 28(5), 755-768.

Bittencourt, L. F., Madeira, E. R. M. and Da Fonseca, N. L. S., 2012. Scheduling in hybrid clouds. *Communications Magazine*. IEEE, 50(9), 42-47.

Devi, U. C. and Anderson, J. H., 2006. Flexible tardiness bounds for sporadic real-time task systems on multiprocessors. In *IPDPS'06, 20th IEEE International Parallel and Distributed Processing Symposium*. IEEE, Rhodes Island, Greece.

Dillon, T., Wu, C. and Chang, E., 2010. Cloud computing: issues and challenges. In *AINA'10, 24th IEEE International Conference on Advanced Information Networking and Applications*. IEEE, Perth, Australia, pp. 27-33.

Hofer, C. N. and Karagiannis, G., 2011. Cloud computing services: taxonomy and comparison. *Journal of Internet Services and Applications*. Springer, 2(2), 81-94.

Karatza, H. D., 2004. Simulation study of multitasking in distributed server systems with variable workload. *Simulation Modelling Practice and Theory*. Elsevier, 12(7), 591-608.

Karatza, H. D., 2006. Scheduling gangs in a distributed system. *International Journal of Simulation: Systems, Science Technology*. UK Simulation Society, 7(1), 15-

22.

Karatza, H. D., 2007. Performance of gang scheduling policies in the presence of critical sporadic jobs in distributed systems. In *SPECTS'07, 2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. SCS, San Diego, CA, pp. 547-554.

Karatza, H. D., 2008. The impact of critical sporadic jobs on gang scheduling performance in distributed systems. *Simulation: Transactions of the Society for Modeling and Simulation International*. Sage Publications, 84(2-3), 89-102.

Karatza, H. D., 2014. Scheduling Jobs with different characteristics in distributed systems. In *CITS'14, 2014 International Conference on Computer, Information and Telecommunication Systems*. IEEE, Jeju Island, South Korea, pp. 1-5.

Kim, K. H., Buyya, R. and Kim, J., 2007. Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In *CCGRID'07, 7th IEEE International Symposium on Cluster Computing and the Grid.* IEEE, Rio de Janeiro, Brazil, pp. 541-548.

Leontyev, H. and Anderson, J. H., 2010. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*. Springer, 44(1-3), 26-71.

Moschakis, I. A. and Karatza, H. D., 2015. A meta-heuristic optimization approach to the scheduling of Bag-of-Tasks applications on heterogeneous Clouds with multi-level arrivals and critical jobs. *Simulation Modelling Practice and Theory*. Elsevier, 57, 1-25.

Papazachos, Z. C. and Karatza, H. D., 2015. Scheduling bags of tasks and gangs in a distributed system. In *CITS'15, 2015 International Conference on Computer, Information and Telecommunication Systems*. IEEE, Gijón, Spain, pp. 1-5.

Rimal, B. P., Choi, E. and Lumb, I., 2009. A taxonomy and survey of cloud computing systems. In *NCM'09, 5th International Joint Conference on INC, IMS and IDC*. IEEE, Seoul, Korea, pp. 44-51.

Stavrinides, G. L. and Karatza, H. D., 2008. In *SPECTS'08, 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. IEEE, Edinburgh, UK, pp. 1-7.

Stavrinides, G. L. and Karatza, H. D., 2009. Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations. *Simulation: Transactions of the Society for Modeling and Simulation International*. Sage Publications, 85(8), 525-536.

Stavrinides, G. L. and Karatza, H. D., 2010. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. *Journal of Systems and Software*. Elsevier, 83(6), 1004-1014.

Stavrinides, G. L. and Karatza, H. D., 2011. Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. *Simulation Modelling Practice and Theory*. Elsevier, 19(1), 540-552.

Stavrinides, G. L. and Karatza, H. D., 2012. Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes. *Future Generation Computer Systems*. Elsevier, 28(7), 977-988.

Stavrinides, G. L. and Karatza, H. D, 2014. The impact of resource heterogeneity on the timeliness of hard real-time complex jobs. In *PETRA'14, 7th International Conference on Pervasive Technologies Related to Assistive Environments*. ACM, Rhodes Island, Greece, pp. 65:1-65:8.

Stavrinides, G. L. and Karatza, H. D., 2015. A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds. In *FiCloud'15, 3rd International Conference on Future Internet of Things and Cloud*. IEEE, Rome, Italy, pp. 231-239.

Streit, A., 2005. Enhancements to the decision process of the self-tuning dynP scheduler. In *JSSPP'05*, *11th Workshop on Job Scheduling Strategies for Parallel Processing.* Springer, Cambridge, MA, pp. 63-80.

Terzopoulos, G. and Karatza, H. D., 2016. Bag-of-Tasks load balancing on power-aware clusters. In *PDP'16, 24th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. IEEE, Heraklion, Crete.

Zhang, Y., Franke, H., Moreira, J. and Sivasubramaniam, A., 2003. An integrated approach to parallel scheduling using gang-scheduling, backfilling and migration. *IEEE Transactions on Parallel and Distributed Systems*. IEEE, 14(3), 236-247.