Performance and Cost evaluation of Gang Scheduling in a Cloud Computing System with Job Migrations and Starvation Handling

Ioannis A. Moschakis
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
Email: imoschak@csd.auth.gr

Helen D. Karatza
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
Email: karatza@csd.auth.gr

***Abstract*—Cloud Computing is an emerging technology in the area of parallel and distributed computing. Clouds consist of a collection of virtualized resources, which include both computational and storage facilities that can be provisioned on demand, depending on the users' needs. Gang Scheduling is an efficient technique for scheduling parallel jobs, already applied in the areas of Grid and Cluster computing. This paper studies the application of Gang Scheduling on a Cloud Computing model, based on the architecture of the Amazon Elastic Compute Cloud (EC2). The study takes into consideration both performance and cost while integrating mechanisms for job migration and handling of job starvation. The number of Virtual Machines (VMs) available at any moment is dynamic and scales according to the demands of the jobs being serviced. The aforementioned model is studied through simulation in order to analyze the performance and overall cost of Gang Scheduling with migrations and starvation handling. Results highlight that this scheduling strategy can be effectively deployed on Clouds, and that cloud platforms can be viable for HPC or high performance enterprise applications.**

***Index Terms*—Cloud Computing, Gang Scheduling, HPC, Virtual Machines**

I. Introduction

Cloud computing refers to the model of computing as a utility, just like water and electricity, where users can have access to vast computational resources based on their requirements. *Infrastructure-as-a-Service (IaaS)* clouds differ from older models, like the Grid, in that they do not pose any restrictions on the software or types of services that are available to the user. In fact Clouds offer the ability to utilize any type of software, either existing or custom-made.

The importance of Cloud computing lies in the opportunity for small companies and organizations to have access to computing infrastructure without the need for prior investment. Therefore ideas that would have required major capital investment, can be implemented on the cloud with minimal costs and reduced risk.

It quickly becomes obvious that Cloud computing can also be applied in the area of HPC. Small institutions and individual scientific teams can now have access to large computational resources not only through the Grid, with all its restrictions and limitations, but also through the Cloud which provides an infrastructure platform with *virtually* infinite resources at a fraction of the cost of maintaining a private cluster, and on a flexible pay-per-use model.

In the core of any distributed system lies its job scheduler which is responsible for the allocation of jobs to servers, or *VMs* in our case. Usually, the scheduling methods implemented in the scheduler aim for better response times and lower slowdowns, by minimizing unnecessary delays [1]. In our model, the scheduler must also tend to the cost of the lease time of VMs aiming for a better cost-to-performance ratio. The modeled system, implements a special case of parallel job scheduling called Gang Scheduling in which jobs consist of tasks that must be scheduled to run simultaneously and concurrently since they are in frequent communication with each other. This requires a one-to-one mapping between tasks and VMs [2], and avoids possible bottlenecks or deadlocks, caused by tasks waiting for input from other tasks that are not running.

Gang scheduling has been extensively studied in the past in the area of distributed and cluster systems [1]–[7]. Karatza in [5]–[7] studied the performance of Adaptive First Come First Serve (AFCFS) and Largest Job First Served (LJFS). Also in [8], [9] Karatza has studied the application of Gang Scheduling along with I/O scheduling and processor failures. Papazachos and Karatza in [1] have studied the application of gang scheduling in two cluster systems. The aforementioned publications applied Gang Scheduling schemes in static systems with a preset amount of servers and single ranges for job sizes.

Elasticity in a Grid modeled system using *Virtual-Computers* as processing units was considered by Nie and Xu in [10]. However this publication focused on non-parallel jobs with deadlines and aimed to maximize utilization while maintaining low failure rates.

Scheduling strategies on Cloud computing platforms have been studied before. Assunção et al. in [11] studied the extension of private clusters through the Cloud. In [12] Sotomayor et al. used the Haizea VM management architecture to study

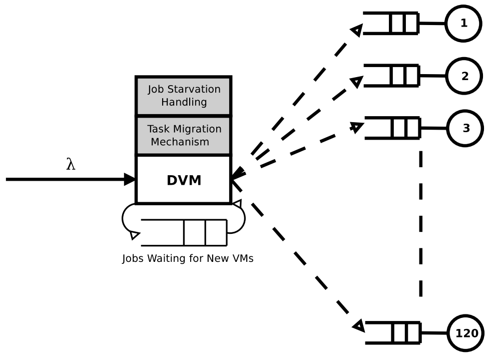


Fig. 1. The system model

Virtual Machine usage in batch scheduling of parallel jobs. In these models jobs' tasks did not need to inter-operate and could be scheduled independently.

In [13], we studied the application of gang scheduling policies in a distributed cloud computing system with dynamic provisioning of VMs. We utilized the two job scheduling algorithms mentioned above AFCFS and LJFS, and conducted simulations for various workloads and multiple job size ranges. The results assessed both the performance and the cost-efficiency of the scheduling algorithms. However, this former work of ours did not consider the migration of job tasks in order the improve response times and reduce fragmentation of jobs, nor did it take into account that in high workloads the adaptive algorithms applied may cause starvation for many of the waiting jobs. In this new study, we integrate both a migration mechanism and a starvation handling system into our model, and compare the effect that these methods have on the overall performance and cost-efficiency of the model. To the best of the authors' knowledge, previous studies have not considered the application of gang scheduling in such a complex model based on a Cloud computing architecture.

The rest of this paper is organized as follows. Section II provides a specification of the system and workload models. Section III describes the strategies that were applied in job scheduling, in the migration mechanism and the starvation handling system. In section IV, we discuss the VM handling methods that have been implemented. Section V presents the metrics used in the assessment of system performance and cost, the parameters of the simulation and the results along with their analysis. Finally section VI offers our conclusive remarks.

II. System & Workload models

The simulation model developed consists of a dynamic cluster of VMs along with a Dispatcher VM (DVM). While the system initially leases no VMs its size can grow or shrink dynamically up to a maximum limit of $VM_{max} = 120$ VMs, a limit described in [14].

Every VM implements its own waiting queue where the DVM is able to dispatch parallel jobs' tasks. The DVM also has a queue for jobs that were unable to be dispatched either due to VM inadequacy at the time of their arrival or due to system load. For simplicity's sake, the DVM is not included in the total VM count. Also, the mechanisms for job migration and starvation handling are considered to be controlled by the DVM.

Communication between VMs is considered *contention-free*, and any communication latencies are *implicitly* included in the jobs' execution time. We do consider, however, *explicit* delays when jobs get delayed in the DVM's queue for the reasons discussed previously.

Furthermore, VMs are considered to belong in the same EC2 instance class and thus have identical characteristics. Though VMs can suffer from performance inequalities, as is true with non-virtualized systems, studies have shown that VMs can provide near homogeneous performance as long as not I/O takes place [15]. Since in our study we do not consider I/O we assume that any overhead due to temporal performance difference is implicitly included in the execution time of jobs.

Gang scheduling requires that the tasks of a job run in parallel [7], and therefore each job requires a number of free VMs equal to its degree of parallelism in order to execute. In our model, degrees of parallelism are random numbers that follow the discrete uniform distribution and fall under two categories of size:

1) **Low-Parallelism Jobs**, with job sizes in the range $[1 \ldots 16]$ with a probability of q.
2) **High-Parallelism Jobs**, with job sizes in the range $[17 \ldots 32]$ with a probability of $1 - q$.

where q is the *job size coefficient* which determines the amount of jobs that belong to the first or the second category.

Thus the Average Job Size (AJS) is computed in the following way:

$$AJS = qE([1 \ldots 16]) + (1 - q)E([17 \ldots 32]) \qquad (1)$$

Where E is the mean value of the discrete uniform distribution for the equivalent range.

The mean inter-arrival time of jobs is exponentially distributed with a mean of $1/\lambda$ and the mean service time is exponentially distributed with a mean of $1/\mu$. Service time and job sizes are not correlated, ergo a low parallelism job can still have a long service time.

Finally studies have shown that context switching in gang scheduling involves high overhead, therefore jobs always execute to completion, and cannot be preempted [16].

III. Scheduling, Migration & Starvation Strategies

A. Job Dispatching

The entry point of the system, as depicted in Fig.1, is the DVM. Jobs with degrees of parallelism less than or equal to the available VMs are dispatched immediately as long as none of the conditions that will be described in section IV are met. For the allocation of VMs to tasks, the DVM applies the *Shortest Queue First (SQF)* algorithm which dispatches the tasks to VMs with the shortest queues.

B. Job Scheduling

Our model applies two of the most commonly used gang scheduling algorithms. Both AFCFS and LJFS have been extensively studied in the area of cluster computing.

1) Adaptive first come first serve: AFCFS tries to schedule jobs whose tasks are in front of their respective queues every time VMs become idle following a departure. If no such job exists, AFCFS tries to schedule jobs that are further down their queues. Because of this way of scheduling AFCFS tends to favor smaller jobs that are easier to schedule and often increases the waiting times of larger jobs.

2) Largest job first served: LJFS on the other hand, gives priority to larger jobs. In every scheduling cycle, LJFS tries to schedule the largest job whose tasks are allocated to idle VMs. This method improves the response time for larger jobs significantly by giving them priority. Also, since larger jobs often leave large enough numbers of VMs free when scheduled, smaller jobs suffer a smaller increase in waiting times in comparison to larger jobs under AFCFS.

C. Job Migration

A common problem that arises in the use of gang scheduling is that, frequently, processors may remain idle while there are waiting tasks in their queues [1]. The implementation of migrations is necessary in order to avoid such fragmentation [17], [18]. The migration process involves the transferring of tasks from the queues of busy VMs to the heads of queues of idle VMs when those are available. Though this process does solve the fragmentation issue mentioned above, it may also incur large overheads to the system [19].

Our system implements certain safeguards to the application of migrations in order to reduce overhead. First of all, migrations are allowed only when the system cannot schedule jobs through the normal route. When that happens the migration system tries to find if there are any jobs that can fit the free VMs. If such jobs exist, the system applies one of the two fitting policies depending on how the simulation is configured in order to select which job to migrate:

1) **First Fit** – The first job, from those that fit, is selected. This method is easier to implement and produces less overhead on average.
2) **Best Fit** – The best fitting job, that is the job with the highest degree of parallelism that fits the free VMs, is selected. This method causes more overhead since it needs to search every job in order to find the best fitting job.

Also for the purposes of this work a migrations monitor, which allows jobs to migrate only every $S_n = 10$ normal job schedulings, has been included to further reduce total migrations.

When the migration process finishes, the migrated job is scheduled for execution in the next scheduling cycle. This happens in order to prevent multiple migrations of the same job, which would introduce more overhead for no reason.

D. Starvation Handling

A reservation system that includes a prioritized queue has been implemented in order to deal with starvation. Normal scheduling of jobs and migrations are paused while this queue contains "starved" jobs. Jobs are considered starved when the *eXpansion Factor (X_{factor})* breaches the "starvation threshold" [20]. The expansion factor is calculated as follows:

$$X_{factor} = \frac{(IWT_j + e_j)}{e_j} \tag{2}$$

Where IWT_j and e_j is the instant waiting time, that is the waiting time up to that point, and execution time of job j respectively.

The selection of the X_{factor} plays an important role in the handling of starved jobs. Moreover, in our model starved jobs can migrate, without being bounded by the above mentioned monitor, therefore the starvation system can lead to a surge in the number of migrations if the X_{factor} selected permits so.

IV. Virtual Machine Handling

The *Cloud* provides users the ability to up-scale or down-scale their available computing resources by requesting more VM instances or by releasing them. This procedure involves a delay, which is derived from the time that the VM setup process will take to create a stated number of VMs. This delay usually is less than 10 minutes per request [14], [21]. In our simulation model these delays were random numbers following a continuous uniform $U(0, 0.2)$ distribution with a mean of 0.1, which is comparable to the one-tenth of the mean service time of job tasks ($1/\mu$) which was 1 for our experiments.

A. Virtual Machine Provisioning

A complex system has been implemented for the addition and removal of VMs from the system. Leasing happens if one of the following conditions is met:

- **Inadequate VMs**, This happens when a job with more tasks than available VMs arrives. The job is queued at the *DVM* until new VMs are provisioned.
- **Overloaded VMs**, At every dispatch the system checks the state of the waiting queues of the VMs and computes the *Average Load Factor* which is equal to:

$$ALF = \frac{\sum_{i=1}^{P_l} t_i}{P_l} \tag{3}$$

  Where t_i is the number of tasks currently waiting at VM i and P_l is the number of VMs leased by the system at that moment.
  Should the ALF surpass a pre-specified load threshold, the system provisions for new VMs equal to the number of tasks of the arriving job and puts it on hold in the DVM queue until such VMs are available.

In any of the above two cases, the system will not try to lease more than the previously mentioned limit VM_{max}. Also after each provision the release mechanism gets paused for 10 arrival cycles in order for new VM queues to fill up.

B. Virtual Machine Releasing

The system may also release VMs under certain circumstances when they are not needed. This operation is particularly important as it affects the cost efficiency of the system. The criteria that deem a VM as releasable are the following:

- The VM is idle and has an empty waiting queue.
- There are no rescheduled jobs in the DVMs queue.
- No migrations are actively scheduled.

All of the above criteria must be met for a VM to be released.

V. Performance & Cost Evaluation

This study focuses both in the evaluation of the performance of the system described in detail above and its cost efficiency.

A. Performance Metrics

The following metrics where employed in the evaluation of performance:

- **Response Time** r_j, of a job j is the time interval between the arrival and the departure of the job. Its average is defined as:

$$RT = \frac{\sum_{j=1}^{N} r_j}{N} \tag{4}$$

  Where N is the total number of jobs.
- **Slowdown** s_j, defined as $s_j = r_j/e_j$, is a metric that compares the delay suffered by a job relative to its service time e_j. Since this metric can be easily affected by very small service times in the denominator we apply the following *bounded* metric [22]:
  Bounded Slowdown:

$$BSLD = \max\left\{\frac{r_j}{\max\{e_j, \tau\}}, 1\right\} \tag{5}$$

  For our experiments τ was set to 10^{-3}
- **Total number of migrations** mig_{tot}, was also evaluated between simulations since the number of migrations will surely affect the overall performance of the system due to the overhead caused.

B. Cost Metrics

Since the Cloud is cost-associative, the system has to maintain a good analogy between response time and cost. Therefore, we integrate *total lease time (LT)* of VMs with *average response time (RT)* in the following metric called Cost-Performance efficiency [13].

$$CPE = D_{LT} + D_{RT} \tag{6}$$

Where D_{LT} is the relative difference in LT between two simulation experiments and D_{RT} is the relative difference between the response times. We must note here that AFCFS was used as a basis for comparison and thus negative values in CPE denote that AFCFS acts better than LJFS.

C. Simulation Parameters

The model described above was implemented using discrete event simulation [23]. Each result represented here is the average of 30 differently instantiated replications of the simulation experiment for each arrival rate (λ), each scheduling algorithm, each migration algorithm and each job size coefficient. Each simulation run was terminated upon completion of 100,000 jobs.

Three different job size coefficients where applied:

- **q=0.25**, with an AJS of 20.5 tasks per job on average.
- **q=0.5**, with an AJS of 16.5 tasks per job on average.
- **q=0.75**, with an AJS of 12.5 tasks per job on average.

These values where selected in order to study the effect of job size on the performance of the system.

The complex structure of the system does not allow for an easy selection of λ values. Through empirical study we have selected values that lead to a system that is able to lease and release VMs dynamically. For each job size coefficient separate arrival rates where studied:

- **For** $q = 0.25$, $\lambda = 1.75, 2.0, 2.25, 2.5$
- **For** $q = 0.5$, $\lambda = 2, 2.25, 2.5, 2.75$
- **For** $q = 0.75$, $\lambda = 2.5, 3.0, 3.5, 4.0$

Also since the expansion factor plays an important role in starvation handling as mentioned above we have tested two separate values for it:

- $X_{factor} = 10$
- $X_{factor} = 20$

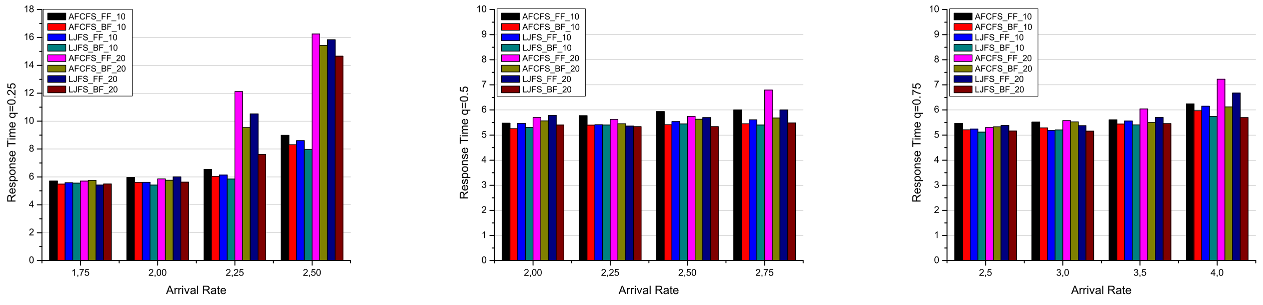
For every mean value, a 95% confidence interval was evaluated. The half-widths of all confidence intervals were less than 5% of their respective values.

D. Results

The results that follow depict the differentiation in performance and cost between the two scheduling algorithms under all different configurations. Response Time is counted in theoretical Time Units (TUs) since the model was simulated with discrete event simulation.

Fig.2a, Fig.2b depict the performance metrics described above, notably RT and BSLD, for $q = 0.25$, $q = 0.50$ and $q = 0.75$ respectively. Each figure shows the results for AFCFS and LJFS in conjunction with First Fit (FF) or Best Fit (BF) and X factors. Fig.3 depicts the number of migrations for each experiment in order to assess the overhead due to migrations. Finally table I lists the *Cost-to-Performance* Efficiency for all parameters.

1) Response Time: It is apparent that the migration system along with starvation handling works very effectively and maintains RT at low levels under low to medium arrival rates notwithstanding the value of the job size coefficient q. Under $q = 0.25$ and $X_{factor} = 20$ in high arrival rates there exists a significant difference in RT. In the same setup LJFS in conjunction with BF offers moderately better results than other combinations of algorithms but BF should be used with caution due to the overhead involved.


(a) Response Time

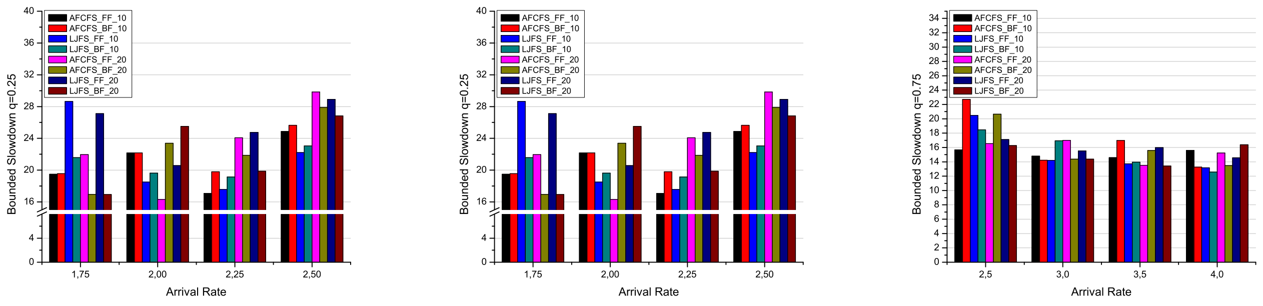

(b) Bounded Slowdown

Fig. 2. Performance Metrics for q=0.25, q=0.50 and q=0.75

As we can see in Fig.3 for $q = 0.25$, $X_{factor} = 20$ allows for significantly less migrations will being on par with $X_{factor} = 10$, in terms of response time, for low arrival rates. In contrast when arrival rates get higher not only the performance for the same X_{factor} degrades a lot, but also the number of migrations is about same as with $X_{factor} = 10$. On the other hand for $q = 0.50$ and $q = 0.75$, $X_{factor} = 20$ manages to maintain a good average response time when compared to $X_{factor} = 10$ while providing up to 40% less migrations. Thus it becomes obvious that a larger X_{factor} may be preferable when dealing with jobs with low to medium parallelism while smaller X factors are better suited do deal with highly parallel jobs even at large arrival rates.

2) Slowdown: Slowdown, as we mentioned above, is a metric that can be easily affected by very small service times and although our metrics are bounded the results can still vary at times.

In Fig.2b the bounded slowdown metric displays a pretty consistent behavior with some outliers, for $q = 0.75$ and $q = 0.50$. These results show that the slowdown incurred by the migration algorithms and starvation handling is at comparable levels for job size coefficients that provide medium to low parallelism jobs. In contrast results for $q = 0.25$ depict higher levels of inconsistency in slowdowns. This can be attributed to the way that migrations and starvation handling functions that may, at times, leave jobs with small service times in the waiting queues for longer periods than expected and thus immediately affecting slowdown.

3) Cost-to-Performance Efficiency: In our previous work [13], which did not include migrations or starvation handling, we came to the conclusion that LJFS is more cost efficient for higher arrival rates under all job size coefficients when compared to AFCFS. Table I shows that these differences almost disappear when migrations and starvation handling comes into play. Different jobs size coefficients and expansion factors seem to have a very small effect, and while the negative values depict that AFCFS is better the difference is practically negligible.

VI. Conclusion

This study, has integrated two important features, job migrations and starvation handling into our previous model [13]. The resulting model was examined through simulation under various workloads, job sizes, migration and starvation handling schemes. Multiple metrics were applied in order to assess both the performance and the cost of the system under all configurations.

The application of migrations and starvation handling had a significant effect on the previous model. Though migrations have balanced the differences in response time their total numbers play an important role in the performance of the system and as such they provided a new basis for comparison. Furthermore the differentiation in expansion factors proved that fine tuning of such mechanisms is required in order to achieve better results under different situations.

In the future we are looking forward to examine new

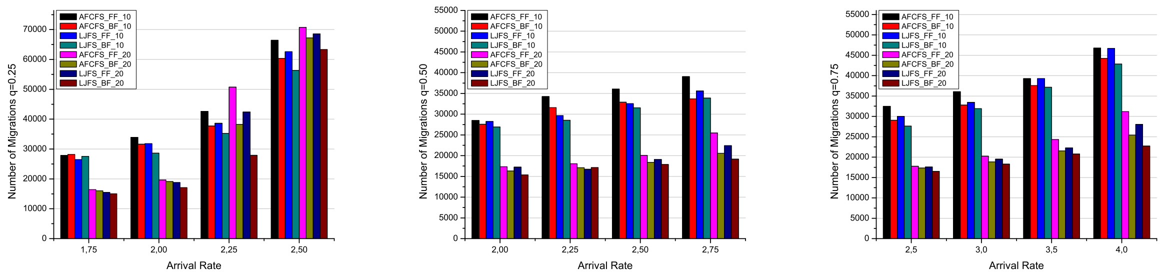


Fig. 3. Number of Migrations for q=0.25, q=0.50 and q=0.75

TABLE I
COST-TO-PERFORMANCE EFFICIENCY TABLE

λ	FF-10	BF-10	FF-20	BF-20
	$q = 0.25$			
1.75	-0.01754	0.01186	-0.05986	-0.05034
2.0	-0.05779	-0.02822	0.01578	-0.02924
2.25	-0.06002	-0.03257	-0.13544	-0.20242
2.5	-0.04291	-0.04251	-0.02578	-0.05216
	$q = 0.50$			
2.0	-0.01108	-0.00255	0.00425	-0.03946
2.25	-0.04043	0.00898	-0.04311	-0.04666
2.5	-0.05804	0.00752	-0.01717	-0.07711
2.75	-0.06253	-0.02212	-0.12085	-0.04923
	$q = 0.75$			
2.5	-0.03205	-0.03452	-0.00519	-0.03172
3.0	-0.04826	-0.02124	-0.05150	-0.07948
3.5	-0.02056	-0.01966	-0.06658	-0.03679
4.0	-0.02614	-0.04835	-0.08809	-0.07641

workload models better suited for Cloud computing. Also the application of these mechanisms in systems with heterogeneous performance should be studied in depth if any conclusions pertaining to a real Cloud system are to be drawn.

REFERENCES

[1] Z. C. Papazachos and H. D. Karatza, "The impact of task service time variability on gang scheduling performance in a two-cluster system," *Simul. Modell. Pract. Theory*, vol. 17, no. 7, pp. 1276–1289, 2009.
[2] ——, "Performance evaluation of bag of gangs scheduling in a heterogeneous distributed system," *The J. of Syst. and Software*, pp. 1–9, 2010.
[3] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration," in *Job Sched. Strateg. for Parallel Process.*, ser. Lect. Notes Comput. Sci., D. Feitelson and L. Rudolph, Eds. Springer Berlin / Heidelberg, 2001, vol. 2221, pp. 133–158.
[4] Y. Wiseman and D. G. Feitelson, "Paired gang scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 6, pp. 581–592, 2003.
[5] H. D. Karatza, "Performance analysis of gang scheduling in a partitionable parallel system," in *Proc. of 2006 Europ. Conf. on Model. and Simul. (ECMS 2006)*, 2006.
[6] ——, "Performance of gang scheduling strategies in a parallel system," *Simul. Modell. Pract. Theory*, vol. 17, no. 2, pp. 430–441, Feb. 2009.
[7] ——, "Scheduling Gangs in a Distributed System," *Int. J. Simul. Syst. Sci. Technol.*, vol. 7, no. 1, pp. 15–22, 2006.
[8] ——, "Gang scheduling and i/o scheduling in a multiprocessor system," in *Proc. of SPECTS'2000, SCS Symp. on Perform. Eval. of Comput. and Telecommun. Syst.* The Society for Modeling & Simulation International, 2000, pp. 245–252.
[9] ——, "Gang scheduling in a distributed system under processor failures and time-varying gang size," in *Proc. of the Ninth IEEE Worksh. on Future Trends of Distrib. Comput. Syst.*, ser. FTDCS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 330–336.
[10] L. Nie and Z. Xu, "An adaptive scheduling mechanism for elastic grid computing," in *Proc. of the 2009 Fifth Int. Conf. on Semant, Knowl. and Grid, SKG '09*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 184–191.
[11] M. D. Assunção, A. Costanzo, and R. Buyya, "A cost-benefit analysis of using cloud computing to extend the capacity of clusters," *Clust. Comput.*, vol. 13, no. 3, pp. 335–347, 2010.
[12] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Resource leasing and the art of suspending virtual machines," in *10th IEEE Int. Conf. on High Perform. Comput. and Commun.* IEEE Computer Society, 2009, pp. 59–68.
[13] I. Moschakis and H. Karatza, "Evaluation of gang scheduling performance and cost in a cloud computing system," *The J. of Supercomput.*, pp. 1–18, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11227-010-0481-4
[14] Amazon elastic compute cloud (amazon ec2). [Online]. Available: http://aws.amazon.com/ec2/
[15] M. A. Murphy, L. Abraham, M. Fenn, and S. Goasguen, "Autonomic clouds on the grid," *J. of Grid Comput.*, vol. 8, no. 1, pp. 1–18, 2010.
[16] A. Hori, H. Tezuka, and Y. Ishikawa, "Highly Efficient Gang Scheduling Implementation," in *Proc. of the 1998 ACM/IEEE conf. on Supercomput.* San Jose, CA: IEEE Computer Society Washington, DC, USA, 1998, pp. 1 – 14.
[17] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," *ACM Comput. Surv.*, vol. 32, no. 3, pp. 241–299, Sep. 2000.
[18] S. Setia, "Trace-driven analysis of migration-based gang scheduling policies for parallel computers," in *Proc. of the int. Conf. on Parallel Process.*, ser. ICPP '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 489–492.
[19] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "The impact of migration on parallel job scheduling for distributed systems," in *Euro-Par 2000 Parallel Process.* Springer, 2000, pp. 242–251.
[20] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, *Selective Reservation Strategies for Backfill Job Scheduling*. Springer Berlin / Heidelberg, 2002, pp. 55–71.
[21] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds : A Berkeley View of Cloud Computing," UC Berkeley, Tech. Rep., 2009.
[22] D. G. Feitelson, "Metrics for parallel job scheduling and their convergence," in *Revis. Pap. from the 7th Int. Worksh. on Job Sched. Strateg. for Parallel Process.*, ser. JSSPP '01. London, UK: Springer-Verlag, 2001, pp. 188–206.
[23] A. M. Law, *Simulation Modeling and Analysis*, 4th ed. McGraw-Hill Higher Education, 2007.