2010 International Conference on Complex, Intelligent and Software Intensive Systems

Job Scheduling in a Distributed System Using Backfilling with Inaccurate Runtime Computations

Sofia K. Dimitriadou
Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
sofiadim@csd.auth.gr

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

***Abstract*—In grid and parallel systems, backfilling has proven to be a very efficient method when parallel jobs are considered. However, it requires a job's runtime to be known in advance, which is not realistic with current technology. Various prediction methods do exist, but none is very accurate. In this study, we examine a grid system where both parallel and sequential jobs require service. Backfilling is used, but an error margin is added to a job's runtime prediction. The impact on system performance is examined and the results are compared with the optimal case of runtimes being known. Two different scheduling techniques are considered and a simulation model is used to evaluate system performance.**

***Keywords*-grid computing; gang scheduling; backfilling; job allocation;**

I. Introduction

Job scheduling for clusters and grid systems plays an essential role in efficiently coordinating large computational resources, and has become increasingly important in recent years. Scheduling can become a challenging task, especially when parallel jobs are considered. The First Come First Served (FCFS) algorithm has proven insufficient and can cause severe fragmentation when resources are not available for large gangs. Backfilling appears to be a promising solution to this problem and has shown to be very effective when gang scheduling is implemented.

Backfilling allows small jobs to initiate before larger queued jobs requiring resources that are currently unavailable. Although this scheduling technique dramatically improves utilization, it also requires that all jobs' service times be known. This information can come from estimates provided by users during job submission or predictions made by the system based on historical data. However, neither of the above methods is very accurate.

This work considers such inaccuracies in runtime predictions and their impact on overall system performance. The result of inaccuracies is that some unfitted jobs will backfill, causing additional delay before a queued gang can begin processing. The resulting system performance is studied using a simulation model, and compared with the optimal case of runtime predictions being exact.

The system under study is a grid system consisting of two sites where both gangs and simple jobs require service. The gangs which enter the system are grid jobs, while the local jobs are simple jobs with only one task, and both job types compete for the same resources. The goal is to provide service to all jobs, while considering the local jobs of higher importance.

A grid job that enters the system will first be dispatched to a specific site by the grid scheduler, and then be further allocated to a processor by the local scheduler. On the other hand, a local job that enters the system arrives directly at the local scheduler. The grid scheduler has its own queue where grid jobs are stored temporarily if specific conditions are not met. The grid jobs that enter the system are parallel jobs (gangs), while the local jobs are simple sequential jobs that require only a single processor for execution. A gang consists of a number of tasks that need to be processed simultaneously, thus each task must be allocated to a different processor. In order for a gang to start execution, all required processors must be available. If backfilling is not implemented, a large gang waiting for resources to become available will block smaller and faster jobs behind it that require execution, causing severe fragmentation in the system. Backfilling allows those jobs to begin and finish execution before the gang, resulting in better overall system performance.

This work extends the study in [1]. While in [1] power-of-2 gang sizes were considered, in this research the gang sizes are uniformly distributed in an integer range from 2 to 13. Furthermore, in this work the system is studied under a different scope. In [1] the predictions of the jobs' service times which were used during backfilling were exact, while in this study an error up to 30% is considered and the impact of this error to the system performance is examined. Another similar scheme with parallel and sequential jobs submitted simultaneously is examined in [2], however it focuses more on the turnaround time of parallel jobs and not local jobs. Backfilling is not used in [2] and the workload model and scheduling policies proposed are fairly different than those discussed in this study. Job scheduling in multi-site systems

978-0-7695-3967-6/10 $26.00 © 2010 IEEE
DOI 10.1109/CISIS.2010.65

is also studied in [3] and [4]. However, neither of these studies considers a model where both local jobs and grid jobs require service. In [3] different scheduling techniques for simple jobs in a 2-level grid system are proposed, while [4] examines the impact of migration in gang scheduling. Previous relevant work also includes job scheduling for distributed systems and computational grids [5-10], gang scheduling [11-15] and backfilling strategies [16-18]. To our knowledge, the study of backfilling with inaccurate runtime predictions under the specified system and workload model does not appear elsewhere in research literature.

The structure of this study is as follows: Section 2 gives a description of the model, Section 3 presents the scheduling policies which are implemented on a grid and local level, section 4 outlines the metrics used to evaluate system performance and analyzes the experiment results, and section 5 provides the conclusion and suggestions for further research.

II. System and workload models

The simulation model considered consists of two sites. Each site is a homogeneous distributed system consisting of a local scheduler and 16 processors, each serving its own queue. We assume that the processors in each site are interconnected through a high speed local area network with negligible communication delays, while the two sites are connected through a wide area network.

The workload consists of two different job types competing for the same resources: local jobs and grid jobs. Therefore, there are three arrival streams in the system: one at the GS (grid jobs) and one inside each of the two sites (local jobs). A local job consists of a single task, while a grid job (gang) consists of a number of parallel tasks that must be processed simultaneously. The reason we chose a synthetic workload over real workload traces is due to real traces being the products of an existing scheduling policy and therefore biased or affected by that policy. Even though they provide a high level of realism when used directly in performance evaluation experiments, their system of origin must resemble the model under study. On the other hand, synthetic workload models offer the convenience of a manageable experimental medium that is free of the idiosyncratic site-specific behaviour of production traces.

In this study, a gang can have 2 to 13 tasks, uniformly distributed. Since this work extends [1], the results will also be compared with the ones in that study, where gang sizes were powers of 2. In [1] gangs could have 2, 4, 8 or 16 tasks uniformly distributed, while in this study the gang sizes follow a uniform distribution with the values ranging from 2 to 13. The gang sizes for this study were chosen so that the average number of tasks per gang in both studies is the same and equal to 7.5.

The mean inter-arrival time of gangs and locals is exponentially distributed with a mean of $1/\lambda_1$ for the locals in site1, $1/\lambda_2$ for the locals in site2 and $1/\lambda_3$ for the gangs, where λ_1, λ_2 and λ_3 are the arrival rates for locals in site1, site2 and gangs, respectively. We assume that the arrival rates in both sites are the same (λ_1=λ_2=λ), and that the arrival rate of grid jobs is much lower than that of local jobs ($\lambda_3 << \lambda$). The service time of a local job or a gang's task is also exponentially distributed with a mean of $1/\mu$.

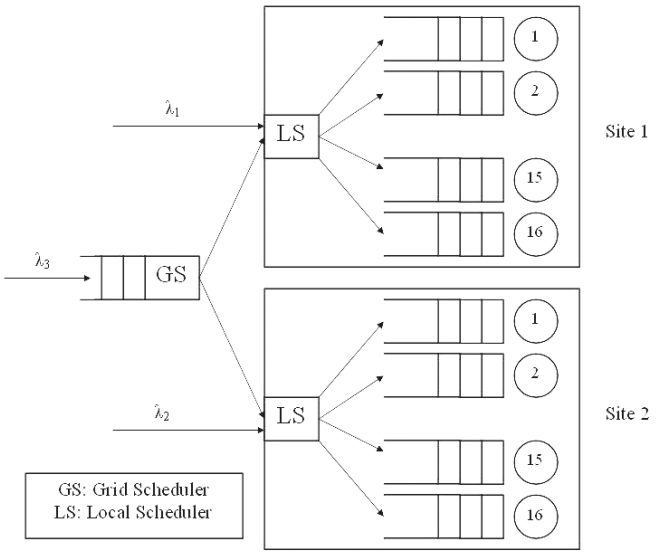

Figure 1. The queuing network model

The communication between the two sites and the GS relies solely on message passing. We assume that it is contention free and therefore the communication time is negligible. However, when a grid job is to be dispatched to both sites, extra coordination is needed and an overhead is added to the job's service time.

In this study, the local jobs have priority over the gangs and it is imperative that their waiting time be minimized. The goal is to provide a quality of service for the locals that will not let the gangs starve. The technique of backfilling is also implemented to help achieve this goal.

The configuration of the model is shown in Figure 1.

III. Scheduling policies

A. Grid level

1) Allocation policies: When a new gang arrives in the system, the GS will determine whether it will be dispatched to the sites or stored in its waiting queue. Each gang's tasks need to be routed to different processors for execution, such that all tasks will be executed simultaneously. Two different gang scheduling approaches are simulated in this study.

Approach 1: In the first scheduling approach, a gang can only be dispatched to a single site. Each site is examined for if it contains enough idle processors to provide service

to all of the gang's tasks. If both sites have enough idle processors to serve the gang, then the GS picks one of them randomly and the gang starts execution immediately. If none of the sites satisfy this condition, then the number of empty queues in each site is computed. If there are enough empty queues in either of the two sites, the gang is scheduled to that site and its tasks are submitted to the empty queues. The gang will start processing as soon as all the processors of those queues become idle. If none of the above conditions are met, then the gang enters the GS's waiting queue.

Approach 2: This approach is an extension of approach 1. The general idea is that a gang can be dispatched to both sites (some of its tasks to site 1 and some of its tasks to site 2) if there are enough idle processors for it to start execution without delay. If the conditions of approach 1 are not met, then the number of idle processors in both sites is computed. If there are enough idle processors in both sites to satisfy the gang's needs, the gang will be routed to both sites and it will start processing immediately. However, an overhead will be added to the gang's service time. If the number of idle processors in site 1 and site 2 is smaller than the number of gang's tasks, then the gang will be put in the GS's waiting queue.

The gang scheduling approaches which were considered in this research have also been studied in [1]. However, in this study the gangs' size distributions are different and the system will be studied under a different scope.

2) Queuing disciplines: Queuing disciplines: When a job (local or gang) exits the system and leaves one or more queues empty, the GS is notified and searches for a gang from its queue that can be scheduled to one or both sites.The algorithm used to determine which gang in the GS's waiting queue will be scheduled next is a modification of the largest gang first algorithm, meaning that priority is given to the gang with the largest number of tasks. The reasoning behind this algorithm is due to large gangs needing more resources, and therefore being more likely to starve. The algorithm works as follows:

1) Each gang in the GS's queue is examined for if it contains less than or equal tasks to the number of empty queues in either of the two sites. If more than one gang satisfies this condition, then the largest one is chosen, and if there is more than one largest gang, then the oldest one is chosen. The chosen gang is scheduled to the site and waits for execution.
2) If a gang was not found during step 2, then the GS checks if there is a gang in its queue with a number of tasks lower than or equal to the number of idle processors in both sites 1 and 2. If such a gang is found, its tasks are routed to both sites and starts processing immediately with a certain overhead. If more than one gang satisfies this condition, then the largest one is chosen, and if there is more than one largest gang, then the oldest one is chosen.
3) Repeat steps 1 and 2 until there is no gang that can be chosen.

Of the approaches studied, approach 2 follows all 3 steps, while approach 1 does not have step 2.

B. Local level

1) Queuing disciplines: When a local job enters a queue, the status of the processor is checked. If the processor is idle and the queue is empty, the job will start processing immediately. If the processor is busy, the job will wait at the end of the queue for its turn to come. Finally, the processor could be idle and the queue not empty. This scenario can occur when a gang is waiting for service, since a gang cannot begin execution unless enough processors are available for all its tasks to be served simultaneously. A large gang may block local jobs behind it in the queue while waiting for sufficient resources to become available.

If the First Come First Served (FCFS) policy is used, the system will suffer from severe fragmentation. Furthermore, the gangs will delay the service of local jobs while the locals are of higher importance. For those reasons, a modified FCFS policy that uses backfilling is applied. This scheduling policy suggests that a local job can take over an idle processor under the condition that delays to the gang in queue are minimal. A job can start execution prior to a gang waiting in the queue if the following condition is met:

$$ServiceTime \leq ElapsedTime + T \quad (1)$$

Where service time is the runtime of the local job that is backfilling and elapsed time is the remaining time until the gang can start execution. The threshold ($T \geq 0$) indicates the maximum time a backfilling local can delay a gang.

When a processor is freed and the queue is not empty, the next job in queue will start execution. If the next job in queue is a gang and it cannot start processing immediately because it needs more resources, the backfilling process begins. The second job in queue will take over the processor if its service time is smaller than the sum of the gang's elapsed time and the threshold. If not, then the same procedure is followed for the next local in queue until a job meeting the backfilling condition is found. If there is no such job, the processor remains idle until the gang can finally start execution.

For this backfilling method to be realized, we need to know the following parameters:

1) The service time of a job.
2) The exact time that all needed resources will be free for the gang to start execution.

The second parameter is not hard to calculate, since the processor can estimate how long it will take for a job which is currently being executed to complete. The first parameter however can never be computed with complete accuracy, and will rely on estimates provided by users when the jobs are submitted or predictions made by the system based on

historical data. Since a job's runtime prediction cannot be estimated accurately in practice, this study will consider errors in such predictions up to a maximum of 30%, in order to simulate a realistic system.

2) Allocation policies: When a local job arrives at one of the two sites, it is dispatched to one of the sixteen processors by the local scheduler. In this study, we use a variation of the shortest queue algorithm to determine to which queue a local job will be scheduled. This algorithm has the following steps:

1) If all processors are busy, the local job will be scheduled to the shortest queue (the queue with the minimum load). If there is more than one, it will be routed to one of them at random.
2) If there are queues with idle processors, the local job will be routed to the one where it can start execution immediately, either because the queue is empty or because the queue is not empty but the following condition is satisfied:

   $$\text{ServiceTime} \leq \text{ElapsedTime} + \text{T}$$

   If this is not possible, the job will be allocated to the shortest queue.

IV. Performance evaluation

A. Performance metrics

In order to evaluate the system's performance, the following metrics will be employed.

The response time r_j of a job *j* is the time interval between this job's arrival into the system until it is completed. Response time includes the waiting time in the queues and the service time in the server.

If *m* is the number of total processed jobs, then the average response time is:

$$RT = \frac{1}{m}\sum_{j=1}^{m} r_j \tag{2}$$

In our system we compute two different response times: the average response time of the gangs and the average response time of the local jobs.

The Slowdown s_j of a job *j* is the response time of that job r_j divided by its service time e_j. This metric is used to measure the delay of a job against its actual runtime, and is defined as follows:

$$s_j = \frac{r_j}{e_j} \tag{3}$$

If *m* is the number of the total processed jobs, then the average slowdown is:

$$SLD = \frac{1}{m}\sum_{j=1}^{m} s_j \tag{4}$$

Table I
NOTATIONS.

P	Number of processors in a cluster
μ	Mean processor service time
$1/\mu$	Mean service rate per processor
λ_3	Mean arrival rate of gangs
λ	Mean arrival rate of local jobs
U	Average processor utilization
RT	Average response time of local jobs
WRT	Average weighted response time of gangs
SLD	Average slowdown of local jobs
$WSLD$	Average weighted slowdown of gangs
T	Threshold
E	Maximum error margin (%) of a jobs' runtime prediction

In this system we compute two different slowdowns: the average slowdown of the gangs and the average slowdown of the local jobs.

The mean response time and the mean slowdown are adequate metrics when they correspond to simple jobs with only one task (local jobs). When parallel jobs (gangs) are being studied, the response time and the slowdown of each gang need to be weighted with its size. This way it is avoided that gangs with a different number of tasks appear to have the same impact on the system. The following weighted metrics are used:

- The average weighted response time *WRT*:

$$WRT = \frac{\sum_{j=1}^{m} p(x_j) r_j}{\sum_{j=1}^{m} p(x_j)} \tag{5}$$

- The average weighted slowdown *WSLD*:

$$WSLD = \frac{\sum_{j=1}^{m} p(x_j) s_j}{\sum_{j=1}^{m} p(x_j)} \tag{6}$$

Where *p(x)* is the number of processors required by job *x*.

Table I contains the parameters used in simulation computations.

B. Input parameters

The queuing network model is simulated with discrete event simulation models using the independent replications method [19]. Each result presented is the average value that is derived from 10 simulation experiments with different seeds of random numbers. Each simulation run is terminated upon the successful completion of 120000 jobs.

In this study we assume that the flow of grid jobs in the system is always being controlled and kept at a standard rate. Therefore, λ_3 is static for all experiments and equal to 0.5, which means that the mean inter-arrival time for the gangs is $1/\lambda_3$=2. However, local jobs' arrival rate can change according to the number of users connecting to the system and the amount of processes they submit to it. In the simulation experiments we set the mean inter-arrival time

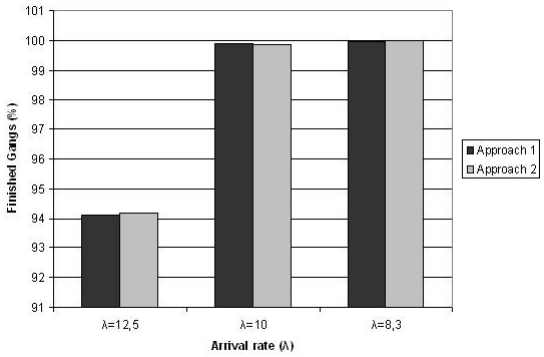

Figure 2. Percentage of finished gangs versus λ for uniform gang sizes

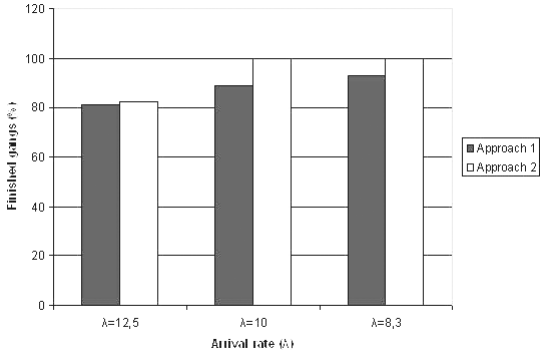

Figure 3. Percentage of finished gangs versus λ for power-of-2 gang sizes

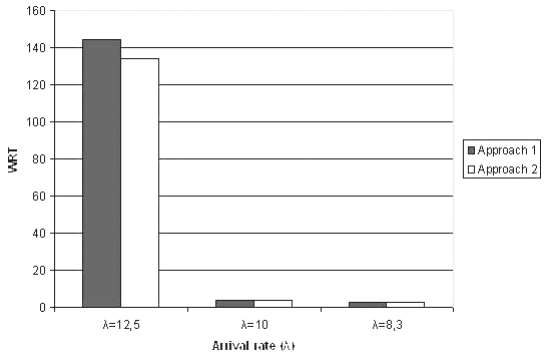

Figure 4. WRT versus λ for uniform gang sizes

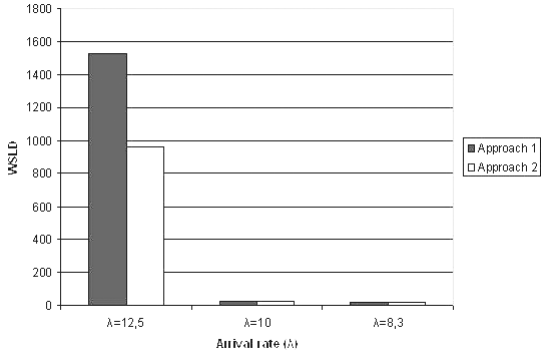

Figure 5. WSLD versus λ for uniform gang sizes

for the locals to $1/\lambda$ = 0.08, 0.1, and 0.12, which correspond respectively to arrival rates: λ = 12.5, 10, and 8.3. We have chosen mean processor service time $1/\mu$ = 1, which implies mean service rate per processor μ = 1. These values were chosen after experimentation, so that the performance of scheduling policies under different loads could be studied without excessive response times.

The threshold (maximum time a backfilling local can delay a gang) is set to T=0 for all experiments and the error percentage of the runtime calculation is set to $E = 0$, $E = 10$ and $E = 30$. For example, if $E = 30$ then a job's runtime prediction can differ from its actual runtime by up to 30%. The overhead for a gang whose tasks have been assigned to processors from various sites is set to 10% of the gang's service time in all experiments.

C. Simulation experiments

In the first set of experiments we keep the error static and equal to $E = 0$ and we examine the impact that the two different gang scheduling approaches have on system performance when the gangs' sizes follow a uniform distribution. The results are also being compared with those of [1] where the gangs' sizes were powers of two.

Figures 2 and 3 depict the percentage of finished gangs as a function of workload when the sizes of gangs are uniform and powers of two, respectively. For uniform gang sizes (Figure 2) we have a higher completion of gangs in all workloads. For medium and high workloads, almost all gangs complete execution in both scheduling approaches.

However, this is not the case when the gang sizes are powers of two (Figure 3). Approach 2 results in the completion of almost 10% more gangs than approach 1, which never completes 100% of gangs even in low workloads. Approach 2 appears to produce much better results when power-of-2 gang sizes are considered and almost the same results as approach 1 for uniform gang sizes. Even if both gang workloads have the same average size (7.5), 25% of power-of-2 gangs entering the system have 16 tasks, while the uniform distribution gangs have a maximum of 13 tasks. Since large gangs are very likely to starve, especially when their tasks can only be routed to one site, this explains low gang completion for approach 1, Figure 3.

Figures 4 and 5 show the mean weighted response time (WRT) and mean weighted slowdown (WSLD) of gangs, as functions of the arrival rate, for uniform gang sizes. For higher workloads, we have higher WRT and WSLD. The figures also show that approach 2 results in lower WRT and WSLD when the workload is high. This is because

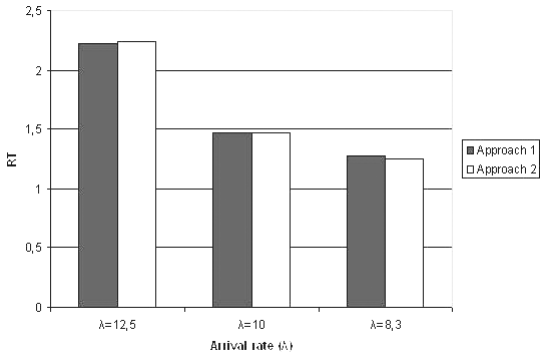

Figure 6. RT versus λ for uniform gang sizes

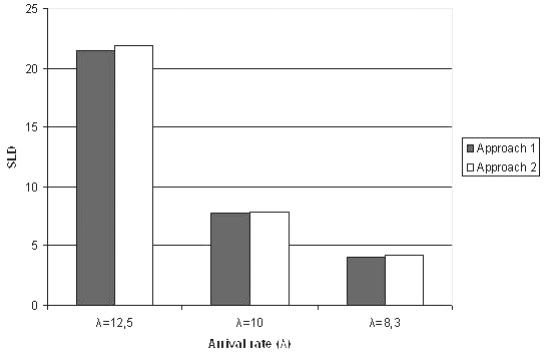

Figure 7. SLD versus λ for uniform gang sizes

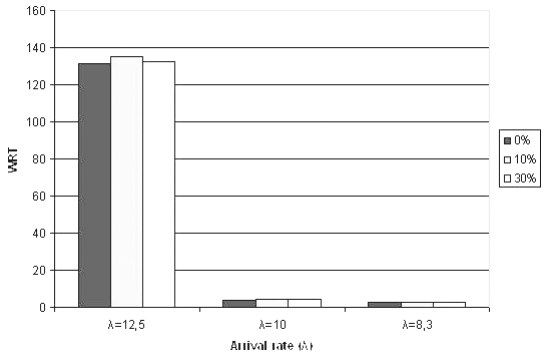

Figure 8. WRT versus λ for uniform gang sizes using approach 1

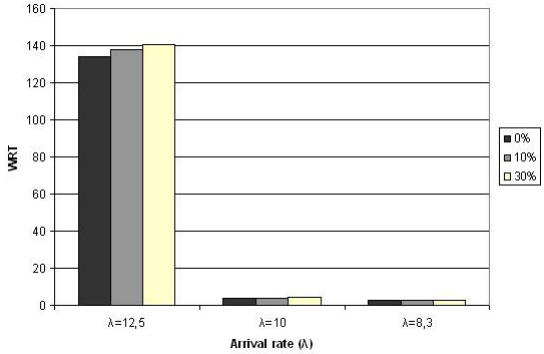

Figure 9. WRT versus λ for uniform gang sizes using approach 2

a gang's tasks can be routed to both sites for execution (oppose to approach 1 where tasks are routed to the same site), which therefore minimizes its waiting time. However, as the workload becomes lower, approaches 1 and 2 produce almost the same results. This leads to the conclusion that gangs are executed as soon as they enter the system no matter which approach is being used, which can also mean that a gangs' tasks are all served in one site, and scheduling them to both sites is not necessary when the workload is low.

Figures 6 and 7 depict the RT and SLD for local jobs as functions of workload, when the gangs' sizes are uniform. Higher workloads result in higher RT and SLD. For $\lambda = 12.5$, RT and SLD are a bit higher for approach 2 compared to approach 1. This is expected since approach 2 serves more gangs in high workloads, which causes local jobs to be delayed slightly. For medium and low workload, the results of both approaches are almost the same. As described earlier, this is because when the workload is low, few gangs need to be scheduled to both sites and the extra steps of approach 2 are rarely used.

Table II shows the mean processor utilization for different workloads. Higher workloads result in higher mean utilization, since processors have more jobs to execute. For the same workload, approach 2 results in slightly higher system utilization. This is expected, since approach 2 assigns gangs to both sites, resulting in the execution of a larger number of gangs and causing the processors to be occupied for a longer period of time.

As shown in the experiments presented above, when uniform gang sizes are considered both approaches produce similar results, especially in medium / low workloads. In high workloads, approach 2 seems to complete a larger percentage of gangs, while keeping a low WRT and WSLD compared to approach 1. However, when power-of-2 tasks are considered (Figure 3 and [1]) approach 2 is the best solution since it completes a higher percentage of gangs in all workloads. This behaviour leads to the conclusion that approach 2 is especially effective when a large percentage of gangs entering the system have many tasks and tends to

Table II
MEAN PROCESSOR UTILIZATION.

	Approach 1	Approach 2
$\lambda = 12.5$	0.87999	0.88034
$\lambda = 10$	0.73785	0.73791
$\lambda = 8.3$	0.63484	0.63490

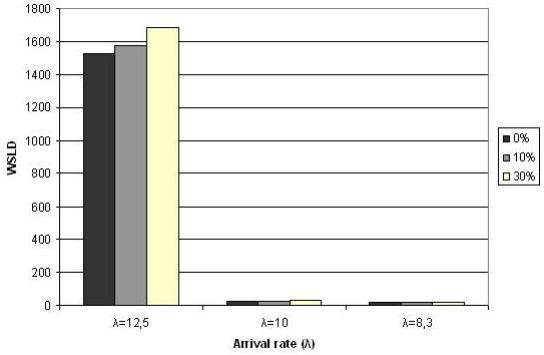

Figure 10. WSLD versus λ for uniform gang sizes using approach 1

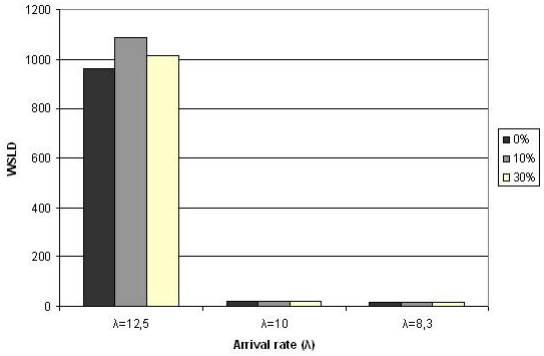

Figure 11. WSLD versus λ for uniform gang sizes using approach 2

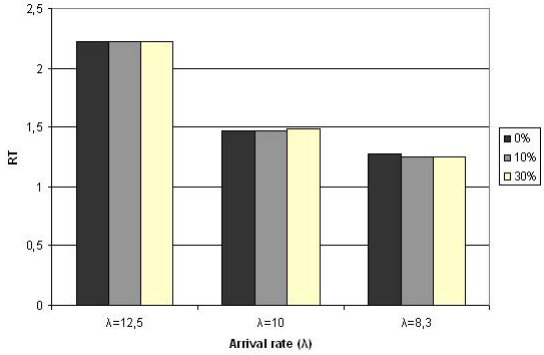

Figure 12. RT versus λ for uniform gang sizes using approach 1

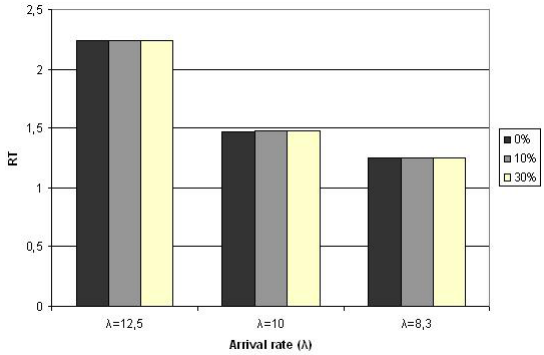

Figure 13. RT versus λ for uniform gang sizes using approach 2

become equal to approach 1 when the gang sizes do not exhibit excessive differences.

In the first set of experiments we assumed that the predictions of the jobs' runtimes which were used for backfilling were exact ($E = 0$). In this set however, we consider a 10-30% error margin in these calculations. The number of tasks a gang has is uniformly distributed in the range of 2 to 13.

Figures 8 and 9 show WRT as functions of arrival rate (λ) for approaches 1 and 2, respectively. For the same workload, the change in WRT for different values of error is very small, especially in low to medium workloads. This can be due to a minimal need for backfilling in these workloads, since the gangs are being executed almost as soon as they enter the system. For high workloads, both approaches result in higher WRT when the error is bigger, while lower WRT is met in the optimal case when there is no error. We can assume that this is caused by low runtime predictions for local jobs, resulting in improper backfilling and additional gang delay. We have similar results for WSLD (Figures 10 and 11).

Figures 12 and 13 depict the response time of local jobs (RT) as functions of the arrival rate for approaches 1 and 2, respectively. Both figures show that the error in runtime predictions of local jobs has little to no effect on their mean response times.

V. Conclusion

This research extends the work in [1]. It studies different scheduling techniques on a system consisting of two sites where grid jobs (gangs) and local jobs compete for the same resources. Two different scheduling approaches were considered and their performance was evaluated. Backfilling was also implemented for gang scheduling in order to avoid fragmentation. An error margin in a job's runtime prediction was considered and its impact on the system was examined. Two sets of experiments were conducted using a simulation model under various workloads.

The experiments indicated that when uniform gang sizes are considered, both proposed approaches produce almost the same results in low to medium workloads, with approach 2 resulting in better performance for high workloads. However, when power-of-2 gang sizes are considered, approach 2 is the best solution in all workloads, since it completes a higher percentage of gangs. Several other experiments were conducted to study the impact on system performance when a job's runtime prediction is not exact. The results show that even when there is error in the predictions, backfilling is an excellent gang scheduling technique for this particular system. Even though predicting a job's runtime with absolute

accuracy is not realistic, error does not significantly affect the overall system performance.

This study could be extended in several ways. For example, we assumed homogeneous sites, when heterogeneity is one of the main characteristics of the grid. Sites with different number of processors could be considered. We also assumed all grid jobs were gangs; in future work, sporadic high priority grid jobs requiring immediate service could also be implemented. Similarly, the workload could be enriched with parallel or critical local jobs.

REFERENCES

[1] S. Dimitriadou and H. D. Karatza,*"Multi-Site Allocation Policies on a Grid and Local Level"*, to appear in the Proceedings of the Fourth International Workshop on Practical Applications of Stochastic Modelling (PASM'09), (Mascots 2009 Workshop), 24 Sept. 2009, Imperial College, London, Elsevier's ENTCS (Electronic Notes in Theoretical Computer Science).

[2] M. Ioannidou and H. D. Karatza, *"Multi-site scheduling with multiple job reservations and forecasting methods"*, Proceedings of the 2006 International Symposium on Parallel and Distributed Processing and Applications (ISPA-06), Sorrento, Italy. Springer, Lecture Notes in Computer Science 4330, pp. 894–903, 2006.

[3] S. Zikos and H. D. Karatza, *"Resource Allocation Strategies in a 2-level Hierarchical Grid System"*, Proceedings of the 41th Annual Simulation Symposium (ANSS), IEEE Computer Society Press, SCS, April 13-16, Ottawa, Canada, pp. 157–174, 2008.

[4] Z. Papazachos and H. D. Karatza, *"Performance Evaluation of Gang Scheduling in a Two-Cluster System with Migrations"*, Evaluation, and Optimization of Ubiquitous Computing and Network Systems (PMEO-UCNS 2009) in Conjunction with IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rome, Italy, May 2009.

[5] C. Franke, U. Schwiegelshohn, R. Yahyapour, *"Job Scheduling for Computational Grids"*, University of Dortmund, Technical Report 0206, 2006.

[6] T. J. Hacker and B.D. Athey, *"A Methodology for Account Management in Grid Computing Environments"*, Proceedings of the 2nd International Workshop on Grid Computing, Springer, Lecture Notes in Computer Science vol. 2242, pp. 133–144, 2001.

[7] H. D. Karatza, *"A Comparative Analysis of Scheduling Policies in a Distributed System Using Simulation"*, International Journal of Simulation: Systems, Science Technology, UK Simulation Society, vol. 1, pp. 12–20, 2000.

[8] R.B. Patel, Neeraj Nehra and V.K. Bhat, *"Distributed Parallel Resource Co-allocation with Load Balancing in Grid Computing"*, International Journal of Computer Science and Network Security (IJCSNS), vol. 7, 282–291, 2007.

[9] A. Tchernykh, J. M. Ramirez, A. Avetisyan, N. Kuzjurin, D. Grushin, S. Zhuk, *"Two Level Job- Scheduling Strategies for a Computational Grid"*, In Parallel Processing and Applied Mathematics, Springer-Verlag, pp. 774–781, 2006.

[10] B. Yagoubi and Y. Slimani, *"Dynamic Load Balancing Strategy for Grid Computing"*, Transactions on Engineering, Computing and Technology, vol. 13, pp. 260–265, 2006.

[11] H. D. Karatza, *"Scheduling Gangs in a Distributed System"*, International Journal of Simulation: Systems, Science Technology, UK Simulation Society, vol. 7, pp. 15–22, 2006.

[12] H. D. Karatza, *"Performance of Gang Scheduling Strategies in a Parallel System"*, Simulation Modelling Practice and Theory, Elsevier, vol. 17, pp. 430–441, 2009.

[13] H. D. Karatza, *"Performance analysis of gang scheduling in a partitionable parallel system"*, in Proc. 20th European Conference on Modelling and Simulation, Sankt Augustin, Germany, pp. 699–704, May 2006.

[14] H. D. Karatza, *"Performance of gang scheduling policies in the presence of critical sporadic jobs in distributed systems"*, in Proc. 2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems-SPECTS 2007, July 16-18, San Diego, CA, pp. 547–554, 2007.

[15] G. Stavrinidis and H. D. Karatza , *"Performance Evaluation of Gang Scheduling in Distributed Real-Time Systems with Possible Software Faults"*, Proceedings of the 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems-SPECTS 2008, June 16-18, Edinburgh, Scotland, UK, pp. 1–7, 2008.

[16] H. D. Karatza, *"A Simulation Model of Backfilling and I/O Scheduling in a Partitionable Parallel System"*, Proceedings of Winter Simulation Conference, ACM, IEEE, SCS, Orlando, Florida, pp. 496–505, December 2000.

[17] P. Wang, Xu Liu, Dan Meng, Jianfeng Zhan, Bibo Tu, *"Dual-Direction Backfilling Algorithm for Job Scheduling"*, HPC Asia conference, Kaohsiung, Taiwan, 2009.

[18] W. Ward, Jr., Carrie L. Mahood, and John E. West, *"Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy"*, 8th International Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 2537, Spinger-Verlag, pp. 88–102, 2002.

[19] A. M. Law and W. D. Kelton, *Simulation modeling and Analysis*, McGraw-Hill, New York, 1991.