Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 261 (2010) 163–179
www.elsevier.com/locate/entcs

Multi-Site Allocation Policies on a Grid and Local Level

Sofia K. Dimitriadou[1] Helen D. Karatza[2]

Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece

Abstract

Efficient job scheduling in computational grids is a challenging task, especially when the workload consists of jobs submitted in a grid and local level. In this study, we consider such a grid system where both local jobs and grid jobs require service. The goal is to maintain a balance between the two competitive job types, in order for every job to be executed in a timely manner. However, local jobs are of higher importance compared to the grid jobs and it is imperative that their waiting time be minimized. Grid jobs are parallel jobs so gang scheduling is implemented, along with various other scheduling techniques in order to improve performance, such as backfilling. A simulation model is considered to evaluate system performance, and experiments are conducted to determine which proposed scheduling policy provides the best results.

Keywords: Grid computing, gang scheduling, backfilling, job allocation.

1 Introduction

The field of grid computing covers a wide scope of concepts and techniques, all of which related to the cooperation of heterogeneous computing resources separated by large distances and with differing administrative domains. Grid computing is defined as "coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations" [1]. The idea behind it is using a large number of geographically distributed resources in order to solve a large problem that could not be solved in any single resource. One of the problems emerging from this concept is resource allocation, and therefore efficient job scheduling. Usually, job scheduling is applied at two levels in computational grids: grid and local. At the grid level, the grid scheduler determines to which system the job will be dispatched and at the local level, the local scheduler allocates the job to the specific resources.

[1] Email: sofiadim@csd.auth.gr
[2] Email: karatza@csd.auth.gr

doi:10.1016/j.entcs.2010.01.011

In a grid system where two different job types exist, job scheduling becomes much more challenging.

In this work, such a system is considered, where both grid jobs and local jobs compete for the same resources. The goal is to provide service to all of them, while the local jobs are considered of higher importance. In order to achieve this goal, various scheduling techniques are applied.

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. Suppose a simple First Come First Served (FCFS) policy was applied, then the local jobs would get blocked by the gangs and would be delayed even if idle processors were available. In order to avoid this kind of fragmentation, the technique of backfilling is applied.

Backfilling allows small jobs to initiate before larger queued jobs which require 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 either estimates provided by users when the jobs are submitted, or predictions made by the system based on historical data [14]. Although none of the above can be completely accurate, in this study we assume that the exact runtime of a job is known. The impact of errors in these assumptions on this particular model will be investigated in future research. However, it has been shown that in general poor estimates of job runtimes do not significantly affect the overall system performance [15].

A similar scheme with parallel and sequential jobs submitted simultaneously is examined in [4], however it focuses more on the turnaround time of parallel jobs and not local jobs. Furthermore, the workload model and scheduling policies proposed are completely different than those discussed in this study. Job scheduling in multisite systems is also studied in [19] and [12]. However, neither of these studies considers a model where both local jobs and grid jobs require service. In [19] different scheduling techniques for simple jobs in a 2-level grid system are proposed, while [12] examines the impact of migration in gang scheduling. Previous relevant work also includes job scheduling for distributed systems and computational grids [2], [3], [5], [13], [15], [18], gang scheduling [6-9], [14] and backfilling strategies [10], [16-17]. To our knowledge, the allocation strategies of grid and local jobs discussed in this study under the specified workload model do 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.

2 System and workload models

The simulation model considered consists of two homogeneous sites. Each site has a local scheduler and 16 processors, while the whole is governed by a Grid Scheduler (GS) that has its own waiting queue. All processors have the same serving capability and they serve their own waiting queue. We assume that the processors in each site are interconnected through a high speed local area network, 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. In this study, the number of tasks a gang can have is a power of 2, which means that each gang can have 2, 4, 8 or 16 tasks (uniformly distributed), and that the mean number of tasks per gang is 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 ($\lambda_3 << \lambda$). We chose exponential inter-arrival times since the Poisson distribution is the most commonly used distribution in the literature for the modeling of job arrivals in both analytical and simulation-based performance evaluation studies of computer systems. Some examples of simulation-based studies where the job arrivals are modeled as a Poisson stream are [3], [4], [6], [7], [12], [13], [14], [19]. The service time of a local job or a gang's task is also exponentially distributed with a mean of $1/\mu$.

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.

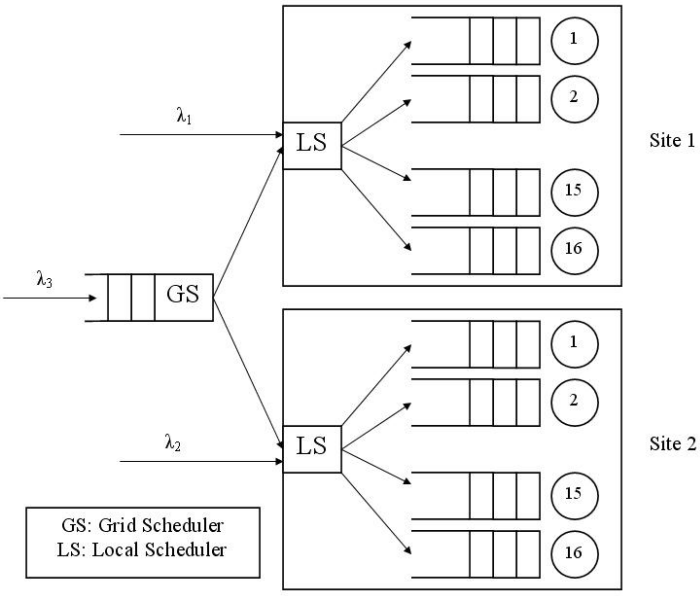

Fig. 1. The queuing network model

3 Scheduling policies

3.1 Grid level

3.1.1 Allocation policies

When a gang enters the system, the Grid Scheduler (GS) will determine whether it will be dispatched to the sites or stored in the waiting queue. Since all the tasks of a gang need to be executed simultaneously, it is obvious that each task must be routed to a different processor. The general idea is that the gang will be routed to the sites only if there are enough empty processors to meet its needs or enough empty queues so that the gang will start processing shortly. In this study, we will be simulating and comparing three different approaches for gang scheduling.

Approach 1: In the first approach, a gang can only be dispatched to a single site. If the number of idle processors in site1 is adequate to satisfy the gang's needs, then the gang will be scheduled to site1 and start processing immediately. If the number of empty processors is not sufficient, the same procedure is followed for site2. If the gang cannot start processing immediately to either of the two sites due to the lack of idle processors, 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: In the second approach, a gang can be dispatched to both sites if there are enough idle processors, which is essentially an extension of the algorithm proposed in approach 1. If the conditions of approach 1 are not met, then the num-

ber 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, some of its tasks to site1 and some to site2, 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 site1 and site2 is smaller than the number of gang's tasks, then the gang will be put in the GS's waiting queue.

Approach 3: In the third approach, a gang can be dispatched to both sites if there are enough empty queues so that it can start execution immediately or in a short period of time. Contrarily to approach 2, where all necessary processors must be available, this approach suggests that a gang can be scheduled to the sites when enough queues are empty, regardless of processor status. Basically, this approach is an extension of approach 2. If the conditions of the second approach are not met, then the number of empty queues in both sites is computed. If it is equal or bigger than the number of the gang's tasks, then the tasks are routed to both sites and will wait in the queues until all processors are idle before starting execution. In this case, an overhead will be added to the gang's service time, since the two sites are assumed to be far enough from each other such that coordination is needed in order for the tasks to be scheduled to both of them.

3.1.2 Queuing disciplines

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. When a job 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 works as follows:

(i) 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 site1. 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.

(ii) The same procedure is repeated for site2.

(iii) If a gang was not found during steps 1 and 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.

(iv) If a gang was not found during the above steps, 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 empty queues in both sites. If such a gang is found, its tasks are routed to both sites 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.

(v) Repeat steps 1 to 5 until there is no gang that can be chosen.

Of the approaches studied, only approach 3 follows all 5 steps. Approach 2 does not have step 4 and approach 1 only has steps 1, 2 and 5.

3.2 Local level

3.2.1 Queuing disciplines

There are three possible scenarios when a local job enters a queue:

(i) The processor is idle, the queue is empty, and the job will be served immediately.

(ii) The processor is busy, and the job will wait at the end of the queue for its turn to come. If the queue is empty, the job will wait at the beginning of the queue and start execution once the processor is free.

(iii) The processor is idle but the queue is not empty.

The last 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. For this to be realized, we make the following assumptions:

(i) The service time of a job can be estimated with accuracy.

(ii) We have knowledge of the exact time that all needed resources will be free and when the gang will start execution.

A job can start execution prior to a gang waiting in the queue if the following condition is met:

$$ServiceTime \leq ElapsedTime + T \tag{1}$$

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.

3.2.2 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 which queue a local job will be scheduled to. This algorithm has the following steps:

(i) 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.

(ii) 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.

4 Performance evaluation

4.1 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:

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

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 1 contains the parameters used in simulation computations.

Table 1
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
WRT	Average weighted response time
SLD	Average slowdown
$WSLD$	Average weighted slowdown
T	Threshold
D_{RT}	Relative (%) decrease in locals' RT when $T > 0$
I_{CG}	Relative (%) increase of completed gangs

4.2 Input parameters

The queuing network model is simulated with discrete event simulation models using the independent replications method [11]. 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 40000 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 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 $\mu = 1$. These values were chosen after experimentation, so that the performance of scheduling policies under different loads could be studied without excessive response times.

4.3 Simulation experiments

In the first set of experiments, we keep the threshold static (T=0) and we examine the impact that the three different gang scheduling approaches have on system performance. 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.

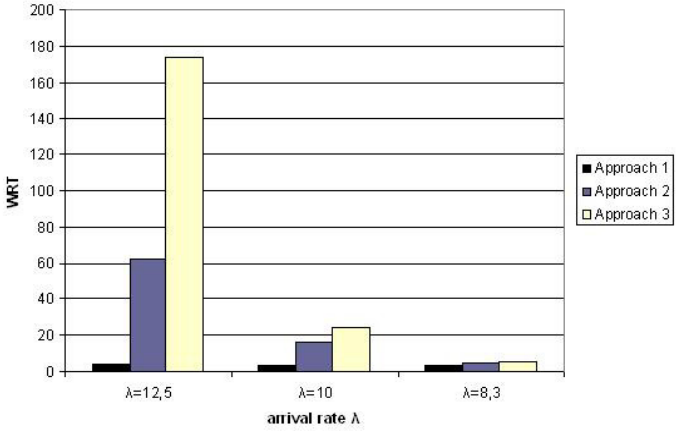

Fig. 2. WRT for gangs versus λ

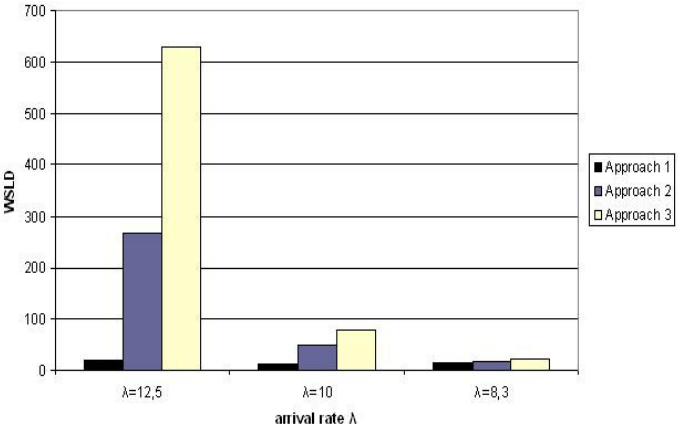

Fig. 3. WSLD for gangs versus λ

Figures 2 and 3 depict the mean weighted response time (WRT) and mean weighted slowdown (WSLD) of gangs, as a function of the arrival rate. For higher workloads, we have higher WRT and WSLD. The figures also show that approaches

2 and 3 give higher WRT and WSLD for the same arrival rate compared to approach 1. This may seem unreasonable, especially considering that approaches 2 and 3 can assign a gang in both sites which would give it a better chance of being executed. The reason the WRT and WSLD are higher in those approaches is illustrated in figure 4, which depicts the percentage of complete gangs, as a function of workload. Approaches 2 and 3 result in the completion of almost 10% more gangs than approach 1, hence higher WRT and WSLD. Approach 1 may cause big gangs to starve, therefore their mean response time and mean slowdown is not computed. Such starvation problems don't occur in approaches 2 and 3, especially in medium workloads. For λ=10 and λ=8.3, 100% of gangs complete execution and exit the system, opposed to 89% and 93% for approach 1. Figure 5 shows the relative (%) increase in gang completion when approaches 2 and 3 are compared to approach 1.

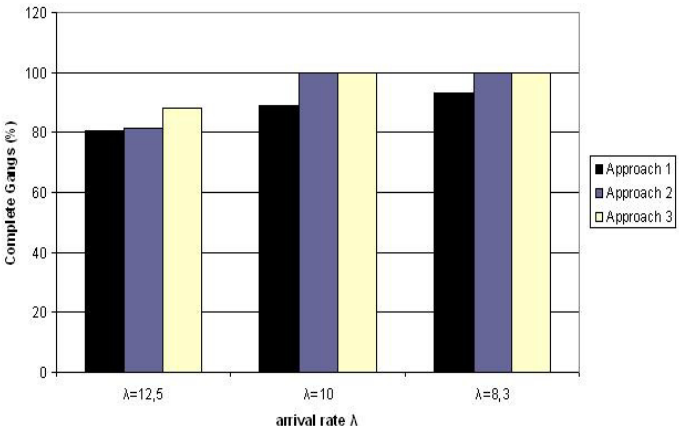

Fig. 4. Percentage of completed gangs versus λ

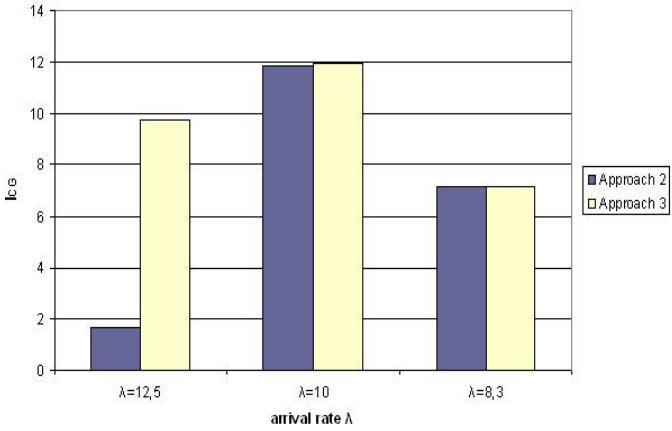

Fig. 5. I_{CG} versus λ, compared to approach 1

Figure 6 shows the local jobs' RT, as a function of arrival rate. For the same workload, approach 3 gives the highest RT for the locals, while the results of ap-

proach 1 and 2 are very similar. The same applies for the locals' mean slowdown (figure 7). Only the SLD for approach 2 and λ=12.5 does not follow this pattern but the difference is insignificant (0.24%) and that can be due to local jobs having a high service time in this particular set of workloads.

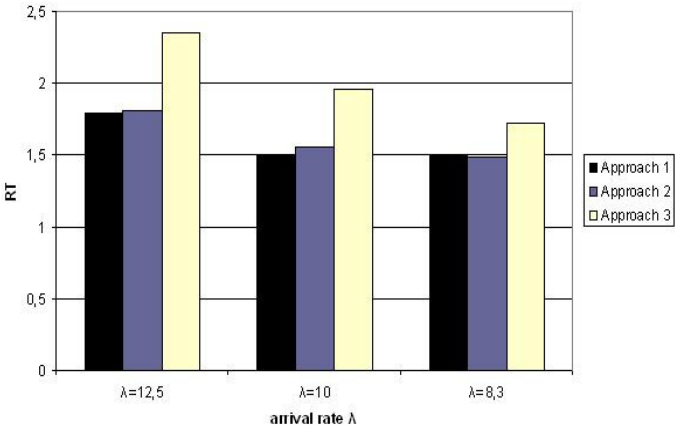

Fig. 6. *RT* for local jobs versus λ

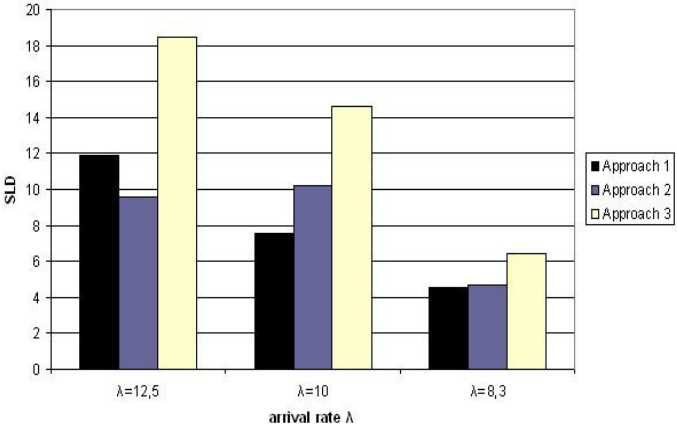

Fig. 7. *SLD* for local jobs versus λ

Table 2 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, approaches that assign gangs to both sites (2 and 3) result in higher system utilization. This is expected, since these two approaches result in the execution of a larger number of gangs, causing the processors to be occupied for a longer period of time.

As shown in the experiments presented above, approach 1 gives the lowest mean response time (RT) and lowest mean slowdown time (SLD) for gangs and locals, but causes gangs to starve (completes only 80% of the gangs in high workloads and 90%

Table 2
Mean processor utilization.

	Approach 1	Approach 2	Approach 3
$\lambda = 12.5$	0.83797	0.84483	0.87845
$\lambda = 10$	0.70227	0.73701	0.75920
$\lambda = 8.3$	0.61289	0.63059	0.64929

in medium/low workloads). Approach 3 completes 90% of gangs in high workloads and 100% in low/medium workloads; however, it gives high average response time and high average slowdown for both gangs and local jobs in all workloads. Approach 2 seems to give the best overall results: it completes a high amount of gangs in all workloads (83% in high workload and 100% in medium/low workload), and results in a satisfying mean response time and mean slowdown for both gangs and local jobs.

In the second set of experiments, we study the impact of the overhead on the system. When a gang is scheduled to both sites a certain overhead is added to the gang's service time. The closer the two sites are to each other, the lower the overhead will be. Normally, lower overhead should produce better results for the gangs' mean response time and slowdown.

For the above set of experiments, we assumed this overhead was 10%. In this set of experiments, two different values for the overhead will be studied: 10% and 5% of the gang's service time. The experiments were conducted using approaches 2 and 3. Approach 1 was not used, since it does not allocate gangs to more than one site and therefore no overhead is added. The arrival rates for local jobs and gangs are the same as the first set of experiments and the threshold is set to T=0.

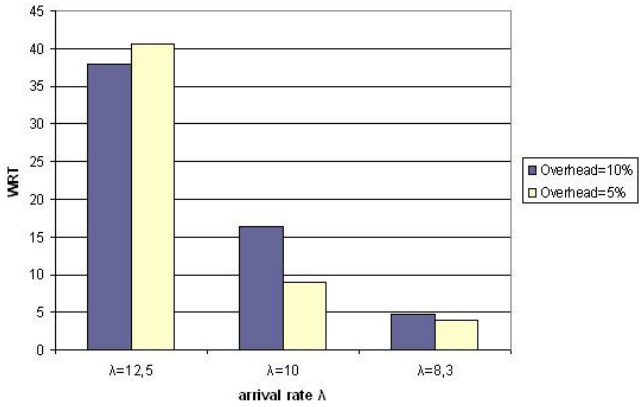

Fig. 8. WRT for gangs versus λ using approach 2

Figures 8 and 9 depict the WRT and WSLD as a function of workload for the two different overhead values when approach 2 is used. As expected, higher overhead results in higher WSLD and WRT in medium and low workloads. However, in a high workload WRT and WSLD will vary largely because so few gangs complete processing. Similar results are produced when approach 3 is applied (figures 10 and

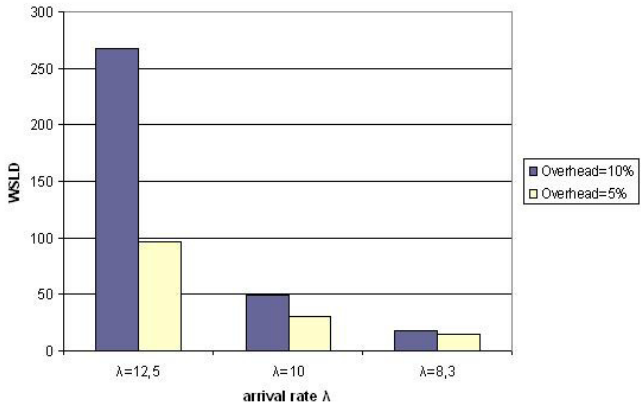

Fig. 9. $WSLD$ for gangs versus λ using approach 2

11).

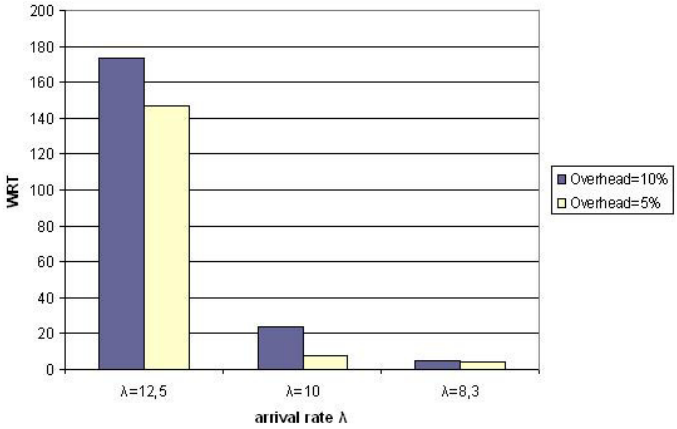

Fig. 10. WRT for gangs versus λ using approach 3

In the third set of experiments, we study the impact of different values of the threshold on the system. For this set of experiments, only approach 2 is used. The arrival rates for local jobs and gangs are the same as the first set of experiments and the threshold is set to T=0, T=0.1 and T=0.15. The overhead is set to 10% of the gang's service time.

Figures 12 and 13 depict the weighted response time (WRT) and the weighted slowdown time (WSLD), as functions of the arrival rate, for three different threshold values. Higher thresholds result in higher WRT and WSLD, which is expected since gangs are delayed by the backfilling locals for a maximum amount of time T. In a less loaded system (λ=8.3), the value of T has minor impact on performance due to there being more idle processors that can serve the locals, which minimizes the need for backfilling.

Figure 14 illustrates the relative decrease in the locals' RT when $T > 0$, as a function of the arrival rate. When $T > 0$, RT is lower for locals, especially in high

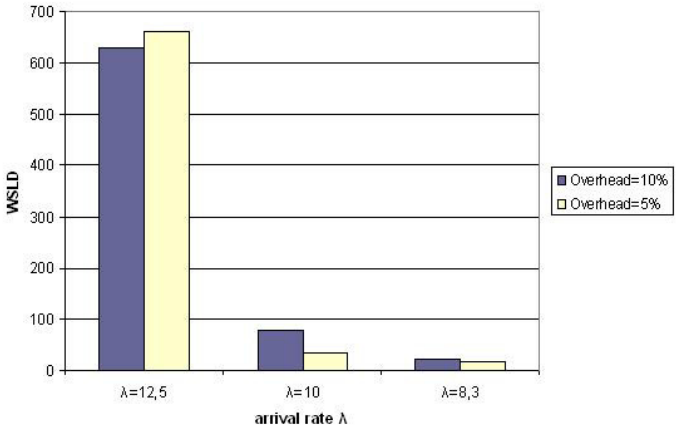

Fig. 11. $WSLD$ for gangs versus λ using approach 3

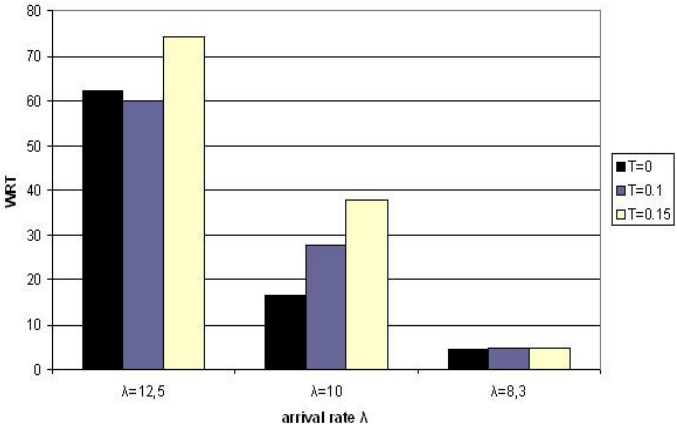

Fig. 12. WRT for gangs versus λ

workloads. For medium and low workloads, there is improvement but it is minor. Overall, keeping T=0 seems to be the best option. When $T > 0$, gangs are slowed significantly, especially in medium/high workloads; this results in a better RT for the locals, but the improvement is small compared to the gangs' high WRT and WSLD.

5 Conclusion and future work

This research studies a system consisting of two sites where grid jobs (gangs) and local jobs compete for the same resources. Different scheduling policies were studied for both gangs and locals and a backfilling method was also implemented in order to avoid fragmentation. Three different scheduling approaches for the gangs were proposed and evaluated. Three sets of experiments were conducted using a simulation model under various workloads.

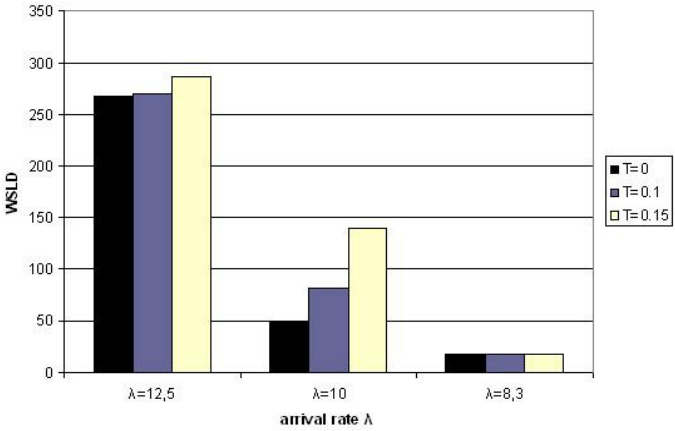

Fig. 13. $WSLD$ for gangs versus λ

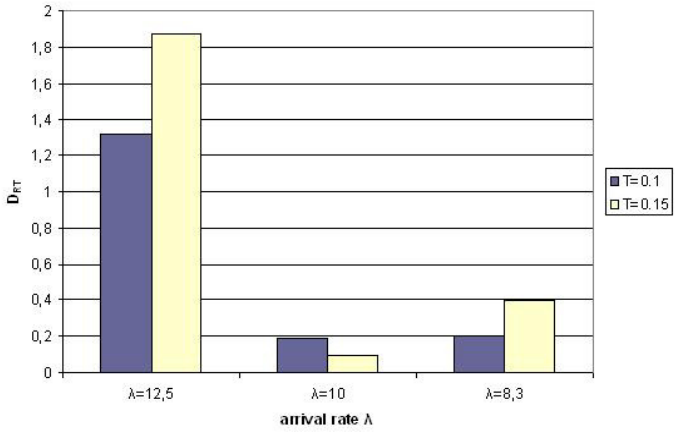

Fig. 14. D_{RT} versus λ

The experiments indicate that approach 2 results in the best overall system performance. It completes a high percentage of gangs (100% in medium/low workloads) without significantly slowing down the locals. Several other experiments were conducted to study the impact on system performance when overhead is added due to gangs being routed to more than one site. These experiments indicated that lower overhead results in better WRT and WSLD for the gangs under almost all workloads. Different values for the threshold of backfilling jobs were also studied and led to the conclusion that T=0 is the best solution, since it results in better WRT and WSLD for the gangs and satisfying RT for the locals.

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. Finally, the model studied only dealt with gangs with sizes as powers of 2. In future work, non-power-of-2 tasks could be considered.

References

[1] Foster, I., C. Kesselman and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International Journal of Supercomputer Applications, **15** (2001).

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

[3] Hacker, T.J. 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, **2242** (2001), 133–144.

[4] Ioannidou, M. 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** (2006), 894–903.

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

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

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

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

[9] Karatza, H. D., *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, 2007, San Diego, CA, 547–554.

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

[11] Law, A. M. and W. D. Kelton, "Simulation modeling and Analysis," McGraw-Hill, New York (1991).

[12] Papazachos, Z. and H. D. Karatza, *Performance Evaluation of Gang Scheduling in a Two-Cluster System with Migrations*, Proceedings of the 8th International Workshop on Performance Modeling, 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 25-29, 2009.

[13] Patel, R.B., 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) **7** (2007), 282–291.

[14] Stavrinidis, G. 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, 1–7.

[15] Tchernykh, A., 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, 2006, 774–781.

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

[17] Ward William, 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** (2002) Spinger-Verlag, 88–102.

[18] Yagoubi, B. and Y. Slimani, *Dynamic Load Balancing Strategy for Grid Computing*, Transactions on Engineering, Computing and Technology **13** (2006), 260–265.

[19] Zikos, S. 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, 2008, Ottawa, Canada, 157–174.