# Resource Allocation Strategies in a 2-level Hierarchical Grid System

Stylianos Zikos and Helen D. Karatza
*Department of Informatics*
*Aristotle University of Thessaloniki*
*54124 Thessaloniki, Greece*
*{szikos, karatza}@ csd.auth.gr*

## Abstract

*Efficient job scheduling in grids is challenging due to the large number of distributed autonomous resources. In this paper we study various resource allocation policies in a 2-level grid system. A simulation model is used to evaluate performance of these policies at the grid level and at the local level. Grid level policies include cases where the grid scheduler uses site information (deferred policy), a random policy, and a combination of the two (hybrid). Simulation results indicate that the hybrid performs better regardless of the local policy.*

## 1. Introduction

In recent years, the trends in parallel processing system design have changed. With the development of Wide Area Networks, centralized supercomputers lose ground to networked distributed systems which share distributed resources in the grid. According to [1] a grid is a system which coordinates resources that are not subject to centralized control.

In computational grids, which are the most common grid form, job scheduling is applied at two levels: grid and local. At grid level, a grid scheduler selects the appropriate systems for jobs, and at local level, local schedulers allocate jobs to specific resources according to a strategy. Grid and local schedulers constitute a scheduling framework which can be centralized or decentralized. The most common decentralized architecture is the hierarchical architecture which includes a grid scheduler, various distributed local schedulers and many resources. With regard to grid schedulers, a classification by centralized and decentralized models can be made. In decentralized models, grid schedulers cooperate to discover a suitable system to serve a job.

A hierarchical architecture includes entities that belong to different levels. For example, in [2] a generic hierarchical tree model with four levels (grid, cluster, site, computing elements) is presented. Hierarchical job scheduling occurs at multiple levels. Hierarchical scheduling strategies for grids are described in [3], where 2-level scheduling strategies are presented. The first level includes job and resource selection strategies, and the second level includes local scheduling strategies. In a hierarchical model, the grid scheduler dispatches jobs to different sites, and the local site schedulers further dispatch the jobs to resources. To route a job to a site, the grid scheduler can use site information for an effective site selection. This information can be based on static or dynamic characteristics of sites [4]. Static characteristics do not change, for example, the number of processors. Dynamic characteristics change over time, for example, the length of local queues. Obtaining real-time global information from sites is costly and leads to high overhead [5]. This is because sites are distributed geographically, and the number may be large, and thus a large amount of communication traffic is required. An improvement could be the use of a fixed update interval [6]. In this case grid scheduler receives dynamic site information only at specific times. In [7], techniques that support efficient task scheduling algorithms in real-time distributed systems were studied, where deadline-based task scheduling and resources allocation were considered jointly.

Previous relevant work includes scheduling in distributed systems [8], [9] and multi-site scheduling [10], where meta-scheduler's decisions are based on predicted load values via time-series analysis. The focus of this paper is on various grid and local resource allocation policies in a 2-level hierarchical grid system. We also evaluate their performance under medium and high workload with the use of the discrete event simulation technique. We examine all the combinations of grid and local policies that we study. Based on the effectiveness evaluation of the use of site information

1080-241X/08 $25.00 © 2008 IEEE
DOI 10.1109/ANSS-41.2008.8

by the grid scheduler, we propose a hybrid policy that could reduce the overhead.

To our knowledge, the combination of grid and local resource allocation policies that we study in a 2-level grid and their effect in system's performance under our workload models does not appear elsewhere in the research literature.

The rest of this paper is organized as follows. In section 2 the model of the system, the scheduling policies, and the metrics used for performance evaluation are described. In section 3 the model's input parameters are outlined and the simulation results are presented and analyzed. Finally, conclusions and suggestions for further research are summarized in section 4.

## 2. Model and methodology

### 2.1. System

A simulation model is used in this paper to study the performance of resource allocation policies. An open queuing network model of a hierarchical 2-level grid system is considered (figure 1). It consists of four sites that are connected through WAN. There is also the Grid Scheduler (GS) that communicates with the sites, so the model is centralized with regard to GS. There is a job arrival stream at the GS (grid jobs) and in that way jobs enter to the system. The GS's task is to dispatch jobs to sites. GS has its own queue, and thus it has the ability to store jobs temporarily if needed. Each site consists of eight processors and a Local Scheduler (LS). A high-speed local network connects all these units of a site. When a job departs from GS, it arrives at the LS of the selected site. There is no queue at LS, so the job is routed instantly to a processor according to a policy. Each processor has its own queue, and a job enters the queue if the processor is busy.

There are no job arrival streams inside the sites, and therefore there are no jobs locally submitted (local jobs). We consider that grid jobs are simple. This means that they cannot be further divided into tasks that can be executed in parallel. We also assume that all jobs can be executed by all processors. The system is homogeneous as all sites have the same number of processors (eight) with the same processing capability (contrary to heterogeneity that characterizes a grid in practice).

The interarrival times of jobs are exponential random variables with mean of $1/\lambda$. Jobs service demands are also exponential random variables with mean of $1/\mu$ (table 1).

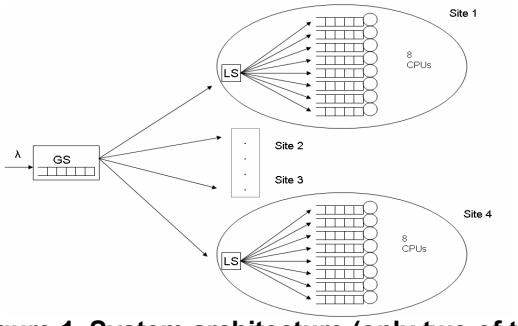


**Figure 1. System architecture (only two of the four sites are shown in figure)**

### 2.2. Policies

With the previously described architecture we made simulation experiments with four different scheduling policies on behalf of the GS and three different scheduling policies on behalf of the LSs.

The GS allocates jobs to sites. The policy determines the way a site is selected for a job. The aim is to achieve a high degree of load balancing among the sites.

***Random GS (R_GS)***

According to Random policy, when there is a job arrival at GS, it randomly selects a site (the probability is the same for all sites) and the job is routed to the LS of this site. In this case, the GS's queue is not used.

***Deferred GS (D_GS)***

The Deferred policy is based on dynamic site load information that the GS receives. The load information is sent to the GS by the LSs. This feedback process occurs at a specified time interval (Allocation Interval). Thus, when there is a job arrival, the GS stores it in the queue and scheduling is deferred. The GS dispatches all jobs in the queue at the end of each Allocation Interval. For each job, the site with the minimum load is selected. We define load as the total number of jobs that exist in a site (the number of jobs in queues plus the number jobs in service). The idea is to take advantage of the feedback information to achieve more efficient load balancing. The drawback is the delay in the GS's queue due to postponement of scheduling. In [11] a deferred scheduling policy in cluster-based servers is proposed. The dispatcher monitors the servers' queues and then routes a job to a server when the number of jobs in the server's queue drops below a predefined threshold. However, in our work it is not required to monitor continuously the load in each site.

***Hybrid GS (H_GS)***

The Hybrid policy is more composite, which combines the two policies mentioned above, Random and Deferred. There still exists the concept of Allocation Interval in which the scheduling is deferred until information from sites becomes available. The

problem with the Deferred policy is the long delay of jobs that arrive at the beginning of the Allocation Interval in GS. Perhaps it's better to route these jobs to a less "good" site with zero delay in the GS's queue. This is the case of Random policy. On the other hand, jobs that arrive in shortly before the end of interval benefit from the best site selection which compensates the delay in queue. This is the Deferred policy part. The question here is when the GS changes policy, from Random to Deferred. A new parameter (A_I%) is introduced which shows the percentage of Allocation Interval in which the Random policy is used. If a job arrives beyond the threshold that A_I% defines, then the GS operates according to Deferred policy. The optimal A_I% value under certain circumstances is examined later in this paper. In figure 2 an example of the GS operation is illustrated, where A_I% = 0.5, implying that half of the time Random policy is used.

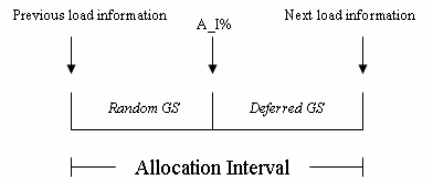


**Figure 2. GS operation when Hybrid_GS policy is used with A_I% = 0.5**

***Real-Time GS (R-T_GS)***

Like Deferred policy, the Real-Time policy is based on information about each site's load. However, in this case scheduling is not deferred and the GS has updated load information at every job arrival. When a job arrives, the GS allocates it to the least loaded site without delay. This scenario is practically unachievable as the overhead from the continuous feedback traffic would be enormous. It's impossible for the GS to know exactly what's happening to a large number of remote sites. However, this Real-Time policy is used in this paper for comparison purposes.

LSs allocate resources (processors) to incoming jobs. The policy determines the way a processor is selected for a job. We consider that each LS has the ability to have up-to-date information about the load in each processor compared to the GS, due to locality.

***Random LS (R_LS)***

According to this policy, each LS randomly selects one of the eight processors to execute a job. The selection probability is the same for all processors.

***Shortest Queue LS (SQ_LS)***

When the Shortest Queue policy is used, a LS uses information about the number of jobs in each local queue and selects the processor with the least number of jobs waiting in queue. In case there are two empty queues, the idle processor is selected. In case that SQ_LS is used and the GS utilizes site information (it depends on the specific GS policy employed), LSs send the GS the number of free processors (with empty queues) in addition to the number of jobs in site. GS selects first the sites with empty queues and then SQ_LS policy guarantees that the job will be allocated to the idle processor. With this optimization, the GS can exploit better the SQ_LS policy.

***2 Random – Shortest Queue LS (2RSQ_LS)***

The 2 Random – Shortest Queue policy is a two-phase policy. Two random processors are selected initially, and then the Shortest Queue policy between these processors is applied. In [12], it is proven that two choices, instead of one, offer exponential improvement in a job's response time in various models that are examined.

We should note here that the FCFS scheduling policy is applied for jobs waiting in queues, both at the GS's queue and at local queues. FCFS ensures certain kind of fairness, does not require in advance information about job execution time, does not require much computational effort, and is easy to implement [13], [14].

## 2.3. Performance metrics

Response time $r_i$ of a job i is the time period from the arrival to the GS to the time service completion of the job. Max response time is the maximum response time of all jobs. The third main parameter is the slowdown metric. Slowdown of a job is the job's response time divided by the job's execution time. If $e_i$ is the execution time of a job i, then the slowdown is defined as follows:

$$s_i = r_i / e_i$$

Table 1 shows the parameters used in simulation computations.

**Table 1. Notations**

| | |
|---|---|
| P | number of processors in system |
| $\lambda$ | mean arrival rate |
| $1/\lambda$ | mean inter-arrival time of jobs |
| $\mu$ | mean service rate |
| $1/\mu$ | mean service demand of jobs |
| A_I | allocation interval |
| A_I% | percentage of A_I (used by the Hybrid GS policy) |
| U | average processor utilization |
| RT | average response time of jobs |
| max RT | maximum RT |
| SLD | average slowdown |
| DSLD | relative decrease in SLD when SQ_LS or 2RSQ_LS policy is employed instead of the R_LS policy |

Let m be the total number of processed jobs. The following metrics used for performance evaluation are defined as follows ([15], [8]):

- The average response time RT:

$$RT = \frac{1}{m} \times \sum_{i=1}^{m} r_i$$

- The average slowdown SLD:

$$SLD = \frac{1}{m} \times \sum_{i=1}^{m} s_i$$

## 3. Simulation results and discussion

### 3.1. Input parameters

The model described above is implemented with discrete event simulation [16]. Each simulation experiment ends when 32000 jobs' executions are completed. Each result presented is the average value that is derived from 10 simulation experiments with different seeds of random numbers. In the given four sites, there are totally 32 processors (P) and the mean service demand of jobs is:

$$1/\mu = 1.$$

If all processors are busy, 32 jobs can be served in one time unit. This implies that we should choose a $\lambda<32$ to maintain the system stability.

We study four cases for the mean job inter-arrival time:

$$1/\lambda = 0.048, 0.043, 0.038, 0.033.$$

The mean arrival rates of jobs are respectively:

$$\lambda = 20.83, 23.26, 26.32, 30.3.$$

The following values for average system utilization are derived theoretically from the chosen $\lambda$ values.

$$U = 65.1\ \%, 73\ \%, 82.2\ \%, 94.7\ \%.$$

### 3.2 Performance evaluation and analysis

The simulation results to be presented next describe performance of the three different LS scheduling policies and the four different GS scheduling policies. Notations of the policies are shown in table 2.

**3.2.1. LS policies performance.** Figure 3 shows that for all arrival rates of jobs the Shortest Queue (SQ) policy yields the lowest average response time, when Random policy is used by GS. The highest average response times are observed with R_LS policy, which is predictable because each selection of a queue is independent and does not take into account the previous state. We also observe that the difference in performance between R_LS and each of SQ_LS and 2RSQ_LS increases with increasing load. It is very interesting to notice that 2RSQ_LS lies between those two curves and is much closer to SQ. Similar results are observed when the GS uses the Real-Time policy (figure 4).

**Table 2. Notations of the policies**

| *Policy* | *Notation* |
|---|---|
| Random GS | R_GS |
| Deferred GS | D_GS |
| Hybrid GS | H_GS |
| Real-Time GS | R-T_GS |
| Random LS | R_LS |
| Shortest Queue LS | SQ_LS |
| 2 Random - Shortest Queue LS | 2RSQ_LS |

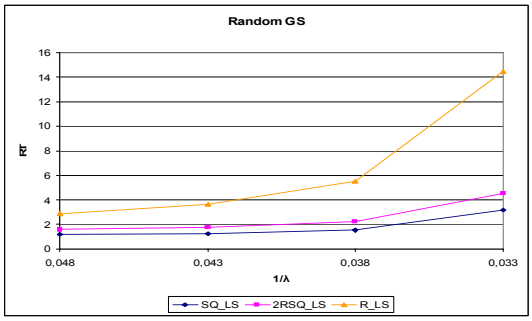


**Figure 3. RT versus 1/λ when R_GS policy is used**

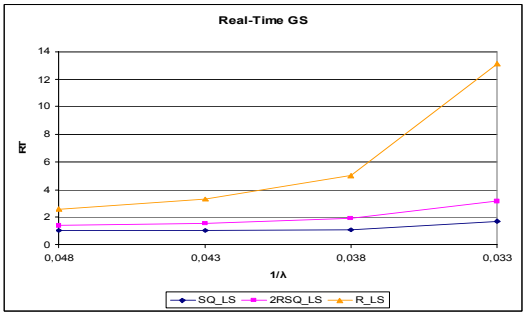


**Figure 4. RT versus 1/λ when R-T_GS policy is used**

Figure 5 illustrates the relative decrease in SLD when SQ_LS and 2RSQ_LS are employed instead of the R_LS policy. SQ_LS method yields the highest DSLD at all arrival rates. This is because SQ_LS performs better than 2RSQ_LS, with lower RT at all arrival rates as we saw above in figure 3. The same behaviour appears in figure 6, where the Real-Time

method is used by the GS. In this case DSLD values are higher, compared to the case where R_GS is used, because of the more effective site selection by GS. However, in both charts (figure 5 and figure 6), we see that all DSLD values are above 60%, which shows the superiority of the two policies (SQ_LS, 2RSQ_LS) over R_LS.

From other simulation experiments that we have conducted, when Deferred and Hybrid policy are applied at GS, the relative performance of the three LS policies does not change.

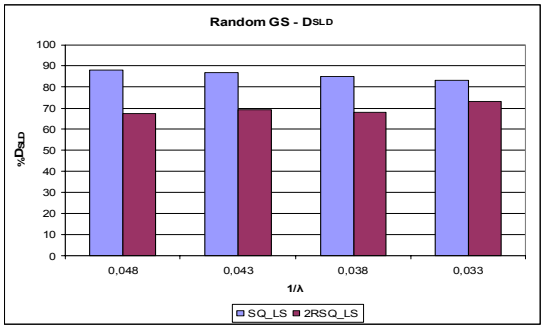


**Figure 5. DSLD(%) versus 1/λ when R_GS policy is used**

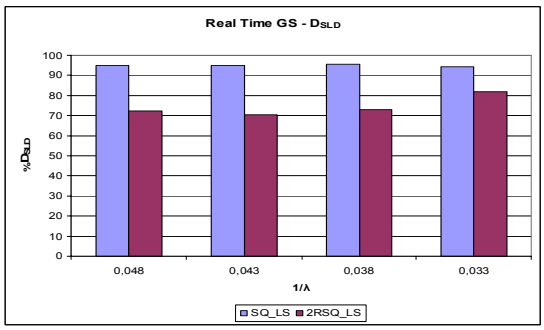


**Figure 6. DSLD(%) versus 1/λ when R-T_GS policy is used**

**3.2.2. GS policies performance.** Figures 7-9 show the RT with regard to arrival rate when R_GS and R-T_GS are used. In figure 7 the Random policy at LS (R_LS) is applied, while in figure 8 and figure 9, the 2RSQ_LS and the SQ_LS are applied, respectively. From these three figures, we can observe that R-T_GS policy performs better (lower RT) in all cases. The result is expected as with R-T_GS, GS has knowledge about each site's load at every job arrival. However, we should be reminded that R-T_GS policy can not be implemented in practice and is used only for comparison purpose. Another observation is that regardless of the LS policy employed, the difference between the two methods increases with increasing load. It seems that the presence of a larger number of jobs affects the R_GS in a larger degree than it affects R-T_GS. This is because there is more efficient load balancing among sites with R-T_GS policy. The difference between the two GS policies is not the same in the three figures; therefore it depends on the LS policy. The use of more effective LS policies favors the GS policy that selects a site based on load information. If there is significant delay in local queues, the benefit from non-probabilistic site selection may not be exploited efficiently. This is the reason the difference between the two policies increases according to this scheme: R_LS → 2RSQ_LS → SQ_LS, as shown in figures 7, 8 and 9, respectively.

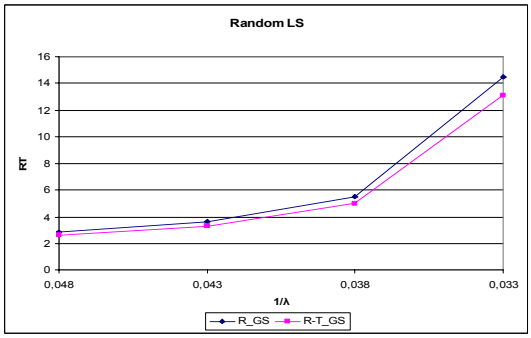


**Figure 7. RT versus 1/λ when R_LS policy is used**

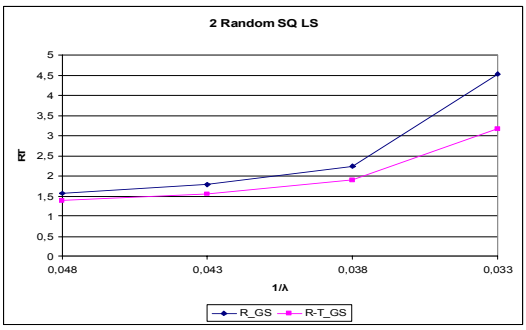


**Figure 8. RT versus 1/λ when 2RSQ_LS policy is used**

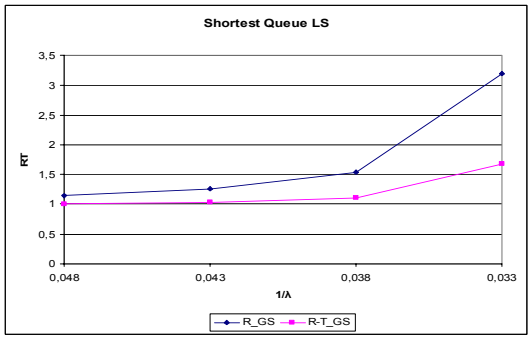


**Figure 9. RT versus 1/λ when SQ_LS policy is used**

Deferred GS policy as described in section 2 is characterized by the Allocation Interval (A_I) parameter. The value of this parameter significantly affects the GS's performance and the system's performance. A high A_I value increases the number of jobs waiting in GS's queue and their delay due to scheduling deferment. On the other hand, a small A_I value eliminates this problem but increases the overhead as load information from sites is required more frequently.

To compare the Deferred GS policy with the Random, through simulation experiments, we found values of A_I that result in *RT with D_GS ≈ RT with R_GS*. These equivalence values are the smallest A_I values for which RT D_GS ≥ RT R_GS. This means that with any A_I that is smaller than equivalence A_I value, D_GS performs better (providing that the arrival rate and LS policy are the same). An example of how the RT is affected with regard to Allocation Interval is shown in figure 10, where the local scheduler's policy is SQ_LS and 1/λ= 0.033. For A_I below 1.2, D_GS performs better than R_GS, and the difference increases for R_GS at higher A_I values.

The results about equivalence A_I are shown in tables 3 and 4. They include the three LS policies and two job arrival rates, with the highest 1/λ=0.033 (table 3) and the lowest 1/λ=0.048 (table 4). As we can observe in table 3, the equivalence A_I values are 1.2, 1.8, 2.5 when SQ_LS, 2RSQ_LS, R_LS are used, respectively. When 1/λ=0.048 (table 4) the equivalence A_I values are 0.3, 0.3, and 0.5, which are much smaller. In this case, a 0.2 (or 0.4) interval is needed for an effective D_GS policy. This means that a large amount of communication overhead is incurred every 0.2 (or 0.4 for the third case) time units for scheduling a small number of jobs. This is because there is relatively little delay in local queues and a longer interval in GS adds extra delay which does not exist with the R_GS policy. Generally a long A_I is desirable for Deferred policy to reduce the communication overhead but with the minimum effect in system's performance.

From the two tables we can observe that equivalence A_I is longer when R_LS is used as compared to SQ_LS and 2RSQ_LS. An explanation is that when R_LS is used the delay in local queues is more significant and a little extra delay in GS is acceptable for a more efficient site selection.

An interesting observation about max RT is that with Deferred policy it is lower than with Random policy, as we can see in the two tables. This is valid for all LS methods and for both job arrival rates. The result is important because the extreme worst case RT values are reduced and the fairness among the jobs increases. The delay of jobs in GS until the end of an Allocation Interval gives the opportunity for the load inside sites to be reduced as there are no new arrivals in the sites awhile.

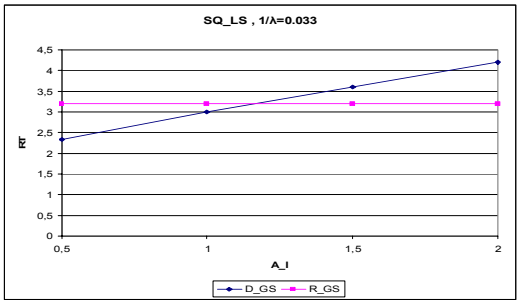


**Figure 10. RT versus A_I when SQ_LS is used and 1/λ=0.033**

**Table 3. R_GS versus D_GS (1/λ= 0.033)**

| SQ_LS | | 2RSQ_LS | | R_LS | |
|---|---|---|---|---|---|
| R_GS | | R_GS | | R_GS | |
| RT | 3.194 | RT | 4.526 | RT | 14.454 |
| max RT | 22.617 | max RT | 25.454 | max RT | 82.84 |
| D_GS | A_I 1.2 | D_GS | A_I 1.8 | D_GS | A_I 2.5 |
| RT | 3.256 | RT | 4.597 | RT | 14.778 |
| max RT | 16.619 | max RT | 20.067 | max RT | 74.151 |

**Table 4. R_GS versus D_GS (1/λ= 0.048)**

| SQ_LS | | 2RSQ_LS | | R_LS | |
|---|---|---|---|---|---|
| R_GS | | R_GS | | R_GS | |
| RT | 1.149 | RT | 1.556 | RT | 2.864 |
| max RT | 11.015 | max RT | 13.397 | max RT | 24.606 |
| D_GS | A_I 0.3 | D_GS | A_I 0.3 | D_GS | A_I 0.5 |
| RT | 1.167 | RT | 1.556 | RT | 2.876 |
| max RT | 10.439 | max RT | 12.703 | max RT | 21.308 |

The proposed Hybrid GS (H_GS) policy schedules jobs with both the Random and Deferred GS methods. It uses the A_I% parameter which determines the percentage of A_I time that Random policy is applied instead of Deferred, as described earlier in this paper. Simulation experiments were conducted to evaluate how H_GS performs as compared to R_GS, D_GS and R-T GS. For D_GS and H_GS the previous equivalence intervals from "Random vs Deferred" experiments were used. The results are presented in the next six figures (11-16), where RT with regard to Hybrid's policy A_I% parameter is shown. The chosen A_I% values are: [0, 0.2, 0.4, 0.6, 0.8, 1]. The Hybrid policy with A_I% = 0 performs exactly like the Deferred because all incoming jobs to GS stay in the queue until load information becomes available. The Hybrid policy performs exactly like the Random when A_I% = 1, because none of the jobs are deferred in GS (in this case A_I is meaningless).

In figures 11-13 the case where mean job inter-arrival time is 0.048 is presented, and in each figure

one of the three LS policies is applied. In figures 14-16 the mean job interarrival time is 0.033 (high load). All figures share certain common characteristics, which are:

1) The RT of three GS policies (D_GS, R_GS and R-T_GS) is constant because these policies are not affected by the A_I% parameter.
2) D_GS and R_GS are very close due to selected A_I values for D_GS (in figure 12 they perform equally, see tables above)

Regarding Hybrid policy, it is obvious that it performs better than D_GS and R_GS for every A_I% between 0.2-0.8. This conclusion is valid for both the 0.048 and 0.033 mean inter-arrival times and for all LS policies. Hybrid's RT decreases with increasing A_I% up to a threshold. After this threshold, RT increases to reach the R_GS. This is due to the following: The combination of Random and Deferred policies allows the GS to benefit from the advantages of both methods: 1) The zero-delay in GS for jobs that would wait the most time if only Deferred used, and 2) The selection of the least loaded sites due to load information. In the beginning the RT decreases with increasing A_I% as Random policy is used more time and less jobs delay in GS's queue. When the A_I% threshold is reached (lowest RT), RT increases with increasing A_I%, because the scheduling is deferred for few jobs, and only these jobs exploit load information from sites. As we observe in figures 11-13, where 1/λ=0.048, the lowest RT appears when 60% of the time Random policy is applied by GS, regardless of LS method (A_I% = 0.6). In figures 14-16, where 1/λ=0.033, the lowest RT appears when A_I% = 0.8. At these A_I% values, Hybrid performs close enough to our reference policy R-T_GS. It is not possible for Hybrid to outperform R-T_GS, as the last one assumes that for every job the least loaded site is selected without any delay.

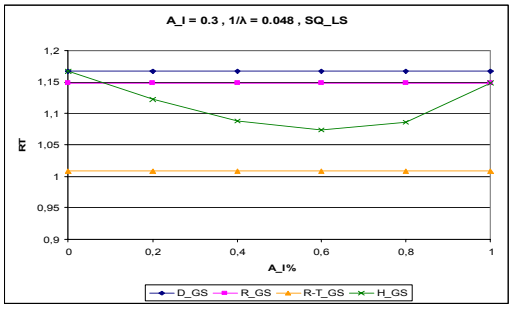


**Figure 11. RT versus A_I% when SQ_LS is used and 1/λ=0.048**

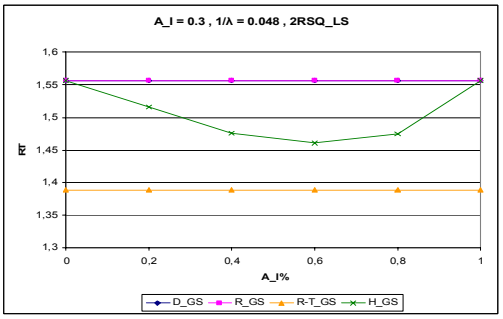


**Figure 12. RT versus A_I% when 2RSQ_LS is used and 1/λ=0.048**

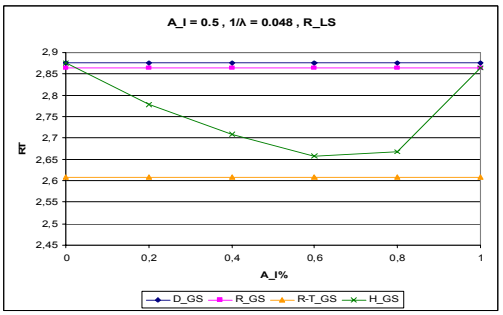


**Figure 13. RT versus A_I% when R_LS is used and 1/λ=0.048**

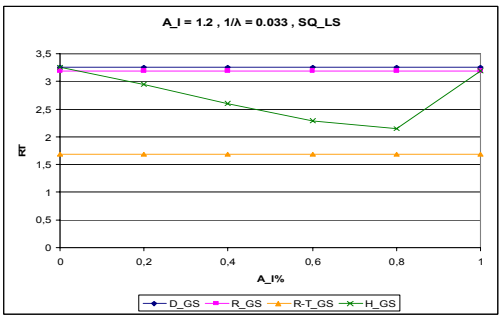


**Figure 14. RT versus A_I% when SQ_LS is used and 1/λ=0.033**

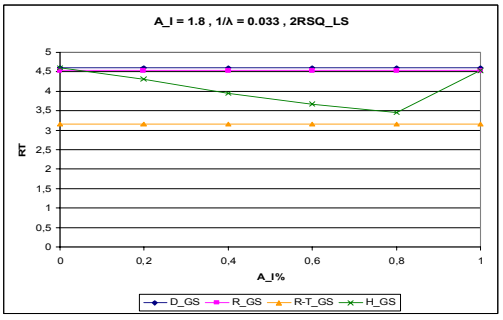


**Figure 15. RT versus A_I% when 2RSQ_LS is used and 1/λ=0.033**

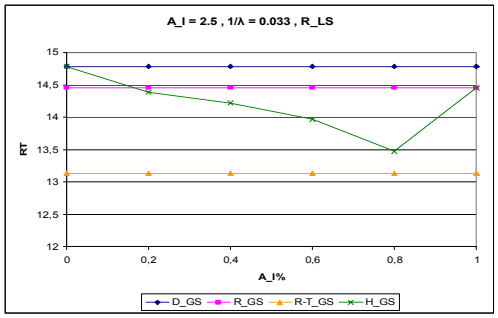


**Figure 16. RT versus A_l% when R_LS is used and 1/λ=0.033**

## 4. Conclusions and further research

This paper studied the performance of four scheduling policies at GS level (R_GS, D_GS, H_GS, R-T_GS) and three scheduling policies at LS level (R_LS, SQ_LS, 2RSQ_LS), in a two-level grid system. Simulation results showed that at LS level, SQ_LS performs best, as expected.

Grid Scheduler's task is to route jobs to sites. R_GS is the simplest policy but it is not effective at high workload. D_GS uses load information to achieve better load distribution, but compared to R_GS it is effective only for high workload due to communication overhead. The proposed H_GS is a combination of the above mentioned two policies which yields a lower RT regardless of the LS method used and the workload. When the optimal R_GS to D_GS ratio is applied, H_GS is approaching our reference R-T_GS policy in performance.

This paper can be extended to the case where the sites are heterogeneous. Heterogeneity, a main characteristic of grids, could be implemented in the model by adding or removing processors, so that sites have different number of processors. In this case, some changes must be done to GS policies presented in this paper to manage heterogeneity, such as the form of load information. Furthermore, the number of sites in the system can be expanded.

## 5. References


1. I. Foster, "What is the Grid? A Three Point Checklist", GridToday, July 2002.
2. B. Yagoubi, Y. Slimani, "Dynamic Load Balancing Strategy for Grid Computing", Transactions on Engineering, Computing and Technology, volume 13, May 2006, pp. 260-265.
3. 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, 2006, pp. 774-781.
4. Y. Cardinale, H. Casanova, "An Evaluation of Job Scheduling Strategies for Divisible Loads on Grid Platforms", In Proceedings of the High Performance Computing & Simulation Conference, Bonn, Germany, May 2006.
5. Y. Patel, J. Darlington, "Allocating QoS-Constrained Workflow-Based Jobs in a Multi-cluster Grid Through Queueing Theory Approach", Parallel and Distributed Processing and Applications, 4th International Symposium, ISPA 2006, Springer-Verlang, 2006, pp. 499-510.
6. E. Caron, V. Garonne, A. Tsaregorodtsev, "Definition, modelling, and simulation of a grid computing scheduling system for high throughput computing", *Future Generation Computer Systems*, Elsevier, Volume 23, Issue 8, 2007, pp. 968-976.
7. Y. Chen, H. Huang, W.T. Tsai, "Scheduling Simulation in a Distributed Wireless Embedded System", *SIMULATION: Transactions of the Society for Modeling and Simulation International*, Vol. 81, Issue 6, June 2005, pp. 425-436.
8. H.D. Karatza, "Performance of Gang Scheduling Policies in the Presence of Critical Sporadic Jobs in Distributed Systems", Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems-SPECTS 2007, San Diego, CA, 2007, pp. 547-554.
9. H.D. Karatza, "Scheduling Gangs in a Distributed System", *International Journal of Simulation: Systems, Science Technology*, UK Simulation Society, Vol. 7, no. 1, (January): 15-22, 2006.
10. M. Ioannidou, H. Karatza, "Multi-site Scheduling with Multiple Job Reservations and Forecasting Methods", In Proceedings of International Symposium on Parallel and Distributed Processing and Applications, volume 4330 of Lecture Notes in Computer Science, Springer, 2006, pp. 894-903.
11. V. Ungureanu, B. Melamed, M. Katehakis, P.G. Bradford, "Deferred Assignment Scheduling in Cluster-Based Servers", *Springer Cluster Computing* 9, 57-65, 2006.
12. M. Mitzenmacher, "On the Analysis of Randomized Load Balancing Schemes", Systems Research Center Technical Note, February, 1998.
13. W. Zhang, C. Albert, M. Hu, "Multisite Co-allocation Algorithms for Computational Grid", In Proceedings of the 3rd High-Performance Grid Computing Workshop (HPGC 2006), Associated with the International Parallel and Distributed Processing Symp. 2006 (IPDPS 2006), IEEE Press, 2006.
14. U. Schwiegelshohn, R. Yahyapour, "Fairness in Parallel Job Scheduling", *Journal of Scheduling*, 3(5):297-320, John Wiley, 2000.
15. A. Streit, "Enhancements to the Decision Process of the Self-Tuning dynP Scheduler", In Job Scheduling Strategies for Parallel Processing book, volume 3277 of Lecture Notes in Computer Science, Springer, May 2005, pp. 63-80.
16. Law, A., D. Kelton, 2006, *Simulation Modelling and Analysis*, McGraw-Hill, New York, 1991.