On Effects of Machine Configurations on Parallel Job Scheduling in Computational Grids

Carsten Ernemann, Computer Engineering Institute, University Dortmund, 44221 Dortmund, Germany
Volker Hamscher, Computer Engineering Institute, University Dortmund, 44221 Dortmund, Germany
Ramin Yahyapour, Computer Engineering Institute, University Dortmund, 44221 Dortmund, Germany
Achim Streit, Paderborn Center for Parallel Computing, Paderborn University, 33102 Paderborn, Germany

Abstract

This paper discusses the influence of the partitioning of parallel resources in a computational grid on the quality of the schedule. To this end, several machine configurations have been simulated with different parallel job workloads which were extracted from real traces. The quality of the parallel job scheduling is measured by the average weighted response or waiting time. The results show that, depending on the communication overhead, the use of partitioned configurations does not necessarily lead to a performance drawback.

1 Introduction

Grid computing is expected to provide easier access to computational resources that are usually limited. Distributed computer systems are joined in a grid environment (see [2, 7]), in which users can submit jobs that are automatically assigned to suitable resources. The idea is similar to meta-computing [15] where the focus is limited to compute resources while grid computing takes a broader approach with for example networks, data, visualization devices etc. as accessible resources [12, 6]. The grid is intended to provide the user with access to locally unavailable resource types. Additionally, there is the expectation that a larger number of resources is available. This is expected to result in a reduction of the average job response time. Also the utilization of the grid computers and the job-throughput is likely to improve due to load-balancing effects between the participating systems.

Now, parallel computing resources are usually not exclusively dedicated to grid computing. Furthermore, they are typically not owned and maintained by the same administrative instance. Research institutes are an example for such resource owners, as well as laboratories and universities. Currently, there is no single large scale grid established as a standard to which resource owners typically join their resources. Moreover, grids are often created as testbeds with a limited number of resources. In this paper we want to discuss the question what influence the number and size of computational resources in a grid have on the quality of service for the user. It is expected that more smaller machines instead of less but larger machines will lead to a drawback for a constant total number of processors. But it is for example unknown if the existence of smaller machines in a grid environment may decrease or increase the schedule quality. In another scenario a company or institution may have the alternative to establish a computer center – with the potential drawback of a performance bottle-neck or in fault-safety – or the setup of two computers at different locations as a grid.

Users are usually interested in a small turn-around time for their jobs. To this end, we examine the influence of different grid configurations on the average waited response time for given workload sets. The workloads are derived from real machine traces to resemble realistic usage.

The execution of jobs in a grid environment is managed by a scheduling system. This system can try to share jobs between the different computers in a grid for load-balancing. Here, jobs are still executed on a single machine. It is likely that grid configurations with smaller machines lead to lower utilization and job-throughput than configurations with larger machines. Besides this scenario the usage of multi-site applications has been theoretically discussed for quite some time [1]. Multi-site computing is the execution of a job in parallel at different sites. This results in a larger number of totally available resources for a single job. The absence of a common grid computing environment which is able to support allocating resources in parallel on remote sites results in a very limited number of real multi-site applications. In addition many user fear a significant adverse effect on the computation time due to the limitations in network bandwidth and latency over wide-area networks. This overhead depends on the communication requirements between the process parts of a particular application. As WAN networks become ever faster, this overhead may decrease over time. For different grid configurations the multi-site execution of jobs may limit the draw-back of smaller machines and improve the schedule quality.

To evaluate the effect of different grid configurations, we examine the usage of multi-site jobs in addition to job sharing. To this end, discrete event simulations on the basis of

workload traces have been executed for sample configurations. The potential drawback is evaluated if smaller computing sites instead of larger participate in a computational grid.

The paper is organized as follows. First, our scheduling model is discussed in the next section. In Section 3 the simulations and their results are presented. The paper ends with a brief conclusion.

2 Model

In this section a description of the scenario which is used in our evaluation is given. After a specification of the underlying models of sites, machines and jobs the scheduling system is introduced.

2.1 Site Model

As mentioned before, we assume a computing grid consisting of independent computing sites. The sites combine their resources and share incoming job submissions in a grid computing environment. Here, jobs can be executed on local *and* remote machines. The computing resources are exclusively committed to grid usage. That is job submissions of all sites are redirected and distributed by a grid scheduler. Only this scheduler controls all grid resources. For a real world application this may be a requirement difficult to fulfill. However, other implementations, where site-autonomy is still maintained, are possible.

2.2 Machine Model

We assume massive parallel processor (MPP) systems as the computing resources. Each site has a single parallel machine that consists of several nodes. Each node has its own processor, memory, disk etc. The nodes inside a machine are connected via a fast interconnection network that does not favor any communication pattern inside the machine [4]. This means a parallel job can be allocated on any subset of nodes of a machine. This model comes reasonably close to real systems like an IBM RS/6000 Scalable Parallel Computer, a Sun Enterprise 10000 or a HPC cluster.

For simplification all nodes in this study are identical. The machines at the different sites only differ in the number of nodes. The existence of different resource types would limit the number of suitable machines for a job. In a real implementation such a preselection is part of grid scheduling and normally executed before the actual scheduling process takes place. After the preselection phase the scheduler can ideally choose from several resources that are suitable for the job request. In this study we neglect this preselection process and focus on the scheduling result. Therefore, it is assumed that all resources are of the same type and all jobs can be executed on all nodes.

The machines support space-sharing and run the jobs in an exclusive fashion. Moreover, the jobs are not preempted nor time-sharing is used. Therefore, once started a job runs until completion. Furthermore, in our study we do not consider the case that a job exceeds its allocated time. After submission a job requests a fixed number of resources that are necessary for starting the job. This number is not changed during the execution of the job. That is jobs are not moldable or malleable [5, 3].

2.3 Job Model

Jobs are submitted by independent users which produces an incoming stream of jobs over time. Therefore, the scheduling problem is an online scenario without any knowledge on future job submissions.

We restrict our simulations on batch jobs, as this job type is dominant on most MPP systems. For interactive jobs there are usually dedicated machine partitions where the effect of the scheduling algorithm is quite limited. In addition interactive jobs are usually executed on local resources.

It is the task of the scheduling system to allocate the jobs to resources and determine the starting time. Then the job is executed without further user interaction. Data management of any files is neglected in this study. In our grid computing scenario, a job can be transmitted and executed at a remote site without any overhead. In a real implementation the transport of data requires additional time. This effect can often be hidden by pre- and postfetching before and after the execution. In this case the resulting overhead is not necessarily part of the scheduling process.

In a grid environment we assume the ability of jobs to run in multi-site mode. That means a job can run in parallel on a subset of nodes belonging to different sites. This allows the execution of large jobs that require more nodes than available on a single machine in the grid environment. The impact of bandwidth and latency has to be considered as wide-area networks are involved. The execution time may increase depending on the communication pattern of the job. If multi-site execution is applied, we address this subject in our simulations by increasing the run time of the affected job.

2.4 Scheduling System

In parallel job scheduling on single parallel machines, simple first-come-first-serve (FCFS) strategies have often been applied. As an advantage, these algorithms provide some kind of fairness (see [14]) and are deterministic for the user. Nevertheless, it can result in poor quality if jobs with large node requirements are submitted. To this end, a strategy called backfilling has become standard on most systems.The backfilling algorithm has been introduced by Lifka [11]. It requires knowledge of the expected job execution time and can be applied to any greedy list schedule. If the next job in the list cannot be started due to a lack of available resources, backfilling tries to find another job in the list which can use the idle resources. But it will not postpone the execution of the next job in the list.

In our scenario for grid computing, the task of scheduling is delegated to a grid scheduler. The local scheduler is only responsible for starting the jobs after an allocation by the grid scheduler. Note, that we use a central grid scheduler for our study. In a real implementation the architecture of the grid scheduler may differ as single central instances usually lead to drawbacks in performance, fail-safety or acceptance of resource users and owners. Nevertheless, distributed architectures can be designed that act like a central grid scheduler.

Two scenarios have been examined in this paper: *job-sharing* between computing sites in a limited grid environment, in addition with *multi-site* computing.

Job Sharing In the *job-sharing* scenario all jobs submitted at any site are delegated to the grid scheduler. In our examination the scheduling algorithms in grid computing consist of two steps. In the first step the machine is selected and in the second step the allocation in time for this machine takes place.

Machine Selection: There are several methods possible for selecting machines. Earlier simulations (presented in [8]) showed the best results for a selection strategy called *Best-Fit*. Here, the machine is chosen on which the job leaves the least number of free resources if started.

Scheduling Algorithm: The backfilling strategy is applied for the single machines as well. This algorithm has shown best results in previous studies [8].

Multi-Site Computing This scenario is similar to *job sharing*: a grid scheduler receives all submitted jobs. Additionally, jobs can now be executed crossing site boundaries (see Figure 1).

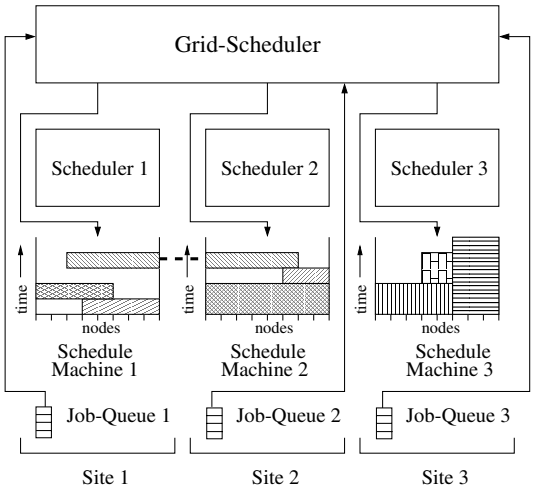

Figure 1 Support for multi-site execution of jobs

There are several strategies possible for multi-site scheduling. For this work we use a scheduler which first after a job submission tries to find a site that has enough free resources for starting the job. If such a machine is not available, the scheduler tries to allocate the jobs on resources from different sites. The number of combined sites is minimized. If there are not enough available resources for a job, it is queued and normal backfilling is applied. Note, that this algorithm is not exactly backfilling as multi-site jobs can surpass queued jobs at submission.

Spreading job parts over different sites usually produces an additional overhead. This overhead is due to the communication over slow networks (e.g. a WAN). Consequently, the overall execution time of the job will increase depending on the communication pattern. For jobs with limited communication demand there is only a small impact. Note, without any penalty for multi-site execution, the grid would behave like a single large computer. In this case, multi-site scheduling will outperform all other scheduling strategies. Within our study we examine the effect of multi-site processing on the schedule quality under the influence of a communication overhead. To this end, the influence of the overhead is modelled by extending the initial execution time r_i^* for a job i that runs on multiple sites:
$r_i^* = (1 + p) \cdot r_i$ with p = 0 .. 60 % in steps of 5%.

3 Evaluation

For the evaluation of the scheduling behavior for different machine structures discrete event simulations were performed. Several workload sets have been examined for the algorithms mentioned earlier.

3.1 Machine Configurations

All configurations use a total of 512 resources. Those resources are partitioned in the various machines as shown in Table 1.

identifier	configuration	max. size	sum
m64	$4 \cdot 64 + 6 \cdot 32 + 8 \cdot 8$	64	512
m64-8	$8 \cdot 64$	64	512
m128	$4 \cdot 128$	128	512
m256	$2 \cdot 256$	256	512
m256-5	$1 \cdot 256 + 4 \cdot 64$	256	512
m384	$1 \cdot 384 + 1 \cdot 64 + 4 \cdot 16$	384	512
m512	$1 \cdot 512$	512	512

Table 1 Resource Configurations

The configurations *m64-8*, *m128* and *m256* represent several sites with equal machines. They are balanced as there is an equal number of resources at each machine. They differ in the number of machines and in the number of resources at

each machine. The configurations *m384* and *m256-5* are examples of a large computing center with several client sites. In the latter configuration, the biggest machine is smaller than the biggest machine within the configuration *m384* and all other machines have an equal size in opposite to *m384*. The configuration *m64* is a cluster of several sites with smaller machines.

Finally, a reference configuration *m512* consists of a single site with one large machine. In this case no grid computing is used and a single scheduler can control the whole machine without any need to split jobs.

3.2 Workload Model

At the moment no real workload is available for grid computing. For our evaluation we derived a suitable workload from real machine traces. These traces have been obtained from the *Cornell Theory Center* and are based on 430 nodes of an IBM RS6000/SP parallel computer. For more details on the traces and the configuration see the description of Hotovy [9]. The workload is available from the standard workload archive [16].

In order to use these traces for the job sharing algorithm it was necessary to modify them to simulate submissions at independent sites with local users. To this end, the jobs from the real traces have been assigned in a round-robin fashion to the different sites. It is typical for many known workloads to favor jobs requiring a power of 2 nodes. The CTC workload shows the same characteristic. The modelling of configurations with smaller machines would put these machines into disadvantage if the number of nodes is not a power of 2. To this end, all our configurations consist of 512 nodes. Nevertheless, the traces consisted of enough workload to keep a sufficient backlog on all systems (see [8]). The backlog is the workload that is queued at any time instance if there are not enough free resources to start the jobs. A sufficient backlog is important as a small or even no backlog indicates that the system is not fully utilized. In this case there is not enough workload available to keep the machines working. Many schedulers, e.g. the mentioned backfilling strategy, require that enough jobs are available for backfilling in order to utilize idle resources. This case usually leads to a bad scheduling quality and unrealistic results.

Over all the quality of a scheduler is highly dependent on the workload. To minimize the risk to achieve singular effects the simulations have been done for 4 workload sets:

- A synthetic probabilistic generated workload on the basis of the CTC traces.
- 3 extracts of the original CTC traces.

The synthetic workload is very similar to the CTC data set [10]. It has been generated to prevent singular effects which occur in real traces in order to preserve the accuracy of the result. Also the usage of 3 extracts of the real traces are used to get information on the consistence of the results for the CTC workload. Each workload set consists of 10000 jobs which corresponds in real time to a period of more than three months.

A problem of such simulations is the handling of wide jobs contained in the original workload traces. The widest job in the CTC traces e.g. requests 336 processing nodes. On one hand these jobs can be used in simulations of multi-site. Here jobs can be split over different sites to get more resources than available at a single site. On the other hand, some of these jobs cannot be started in simulations of the job sharing scenario.

To permit a valid comparison of the simulation results, no job must be neglected. Therefore we assume that the corresponding workloads of wide jobs are still generated at single sites, but are split up into several parts with maximum 64 nodes to allow their execution on all examined configurations. To allow the comparison of different scenarios for job sharing the following modifications have been applied to each forementioned workload.
Workload Modification:

1. Wide jobs split in parts of 64 nodes,
2. Wide jobs unchanged.

The workloads with these modification were executed in both scenarios. Note, that all of these modifications do not alter the overall amount of workload. The simulations allow the examination of the impact caused by wider multi-site jobs on the schedule.

The used workloads will be summarized in Table 2 and an identifier introduced for each workload.

identifier	description
10_20k_org	An extract of the original CTC traces from job 10000 to 20000.
10_20k_max64	The workload 10_20k_org split into jobs with at most 64 processors.
30_40k_org	An extract of the original CTC traces from job 10000 to 40000.
30_40k_max64	The workload 30_40k_org split into jobs with at most 64 processors.
60_70k_org	An extract of the original CTC traces from job 60000 to 70000.
60_70k_max64	The workload 60_70k_org split into jobs with at most 64 processors.
syn_org	The synthetically generated workload derived from the CTC workload traces.
syn_max64	The workload syn_org split into jobs with at most 64 processors.

Table 2 The used workloads

3.3 Results

The simulation results show that configurations with equal sized machines provide significant better scheduling results than machine configurations that are not balanced. Machine configurations with a small number of machines produce better scheduling results under the precondition that each configuration has the same amount of resources in the sum.

As a measure the average weighted response time and the average weighted wait time are used in this study. The response time of each job is the difference between the completion time and the submission time. The response time of each job is weighted by its resource consumption. The average weighted response time is the sum of all weighted response times divided by the number of all jobs. The wait time of each job is the difference between the start time and the submission time. The weights are defined the same way as for the average weighted response time.

Average Response Time weighted by Width: AWRT =

$$\frac{\sum_{j \in Jobs} (j.requestedResources \cdot (j.endTime - j.submitTime))}{\sum_{j \in Jobs} j.requestedResources}$$

Average Wait Time weighted by Width: AWWT =

$$\frac{\sum_{j \in Jobs} (j.requestedResources \cdot (j.startTime - j.submitTime))}{\sum_{j \in Jobs} j.requestedResources}$$

Note that the mentioned weights prevent any prioritization of small over wider jobs in regard to the average weighted response and average weighted wait time if no resources are left idle [13]. The average weighted response time is a mean for the schedule quality from the user perspective. A shorter AWRT indicates that the users have to wait less for the completion of their jobs. A shorter AWWT indicates that the jobs have to wait less time before they are started. Note that the average weighted wait time should not be too small because this would indicate that the backlog is very small and the jobs are calculated nearly directly after submission.

As an example for the scheduling quality the average weighted response time for the workload 60_70k_org and all resource configurations is given in Figure 2. The other workloads show a similar behavior. Several parameter setting for the increase of the corresponding run time for multi-site jobs were examined. The AWRT for the machine configuration m512 is constant for all parameters, due to the fact that no multi-site scheduling is applied in this configuration. As it can be seen in Figure 2 the AWRT decreases from m64-8 over m128 to m256. All of these machine configurations have equal sized machines but the size of the machines increases between the configurations. The AWRT decreases from machine configuration m64 to m64-8, because the configuration m64-8 consists of more larger machines than m64. As more jobs are executed in multi-site mode in configurations with a higher number of smaller machines, an increase of the communication overhead (p) has a higher impact on the AWRT. The same effect can be observed between m256 and m256-5. Here the configuration m256-5 is not balanced and has some smaller machines, which turns out to be a disadvantage. It can be concluded that there is no hard rule for the advantage of equal sized nor of biggest machines. This will highly depend on the characteristic of the workload as can be seen in the comparison between m384 and m256. In this example m384 outperforms m256 for a communication overhead up to 45%.

In Figure 3 the average weighted response time is shown related to the average weighted response time of configuration m512. This underlines the above statements more detailed.

The AWRTs of the configurations m64 and m64-8 are almost similar for values of the multi-site parameter p between 0% and 35%. If the overhead exceeds 35% the AWRT of configuration m64 increases dramatically and is at least 25% bigger than the AWRT for configuration m64-8.

The decreasing AWRT from m64-8 to m128 to m256 can be evaluated with at least 20% for each step for overhead parameters over 35%. The difference of the AWRT for parameters under 35% is not significant.

The comparison between configuration m256 and m256-5 results in a similar conclusion. The AWRT increases for parameters over 45% with at least 25%.

The increase of the AWRT results from two effects. First of all, the whole workload increases, because all multi-site jobs have a longer execution time. Second, the number of multi-site jobs increases. Table 3 shows the number of multi-site jobs for machine configuration m128 for different workloads and different multi-site parameters. For all workloads the number of multi-site jobs increases. This process is not continuously, but the difference between the number of multi-site jobs for p=0% and p=60% is always at least 30%.

This behavior results from the scheduling policy to schedule all jobs as soon as possible after submission. Because of an increased execution time of all multi-site jobs the number of free time slots within the schedule decreases and the probability of free time slots within one machine decreases as well. Therefore jobs can only be started as soon as possible if free time slots from different machines are combined.

Table 4 indicates that the increasing number of multi-site jobs corresponds with an increasing part of the squashed area [1] of all multi-site jobs related to the squashed area of the whole workload. Between parameter p=0% and p=60% this parts increases at least 50%.

Tables 3 and 4 also indicate that jobs running in multi-site mode are in majority bigger jobs. This can be concluded because about 5% to 10% of all jobs are responsible for about 20% to 40% of the whole workload depending on the

[1] The squashed area is defined as:
$\sum_{j \in Jobs} (j.requestedResources \cdot j.runTime)$.

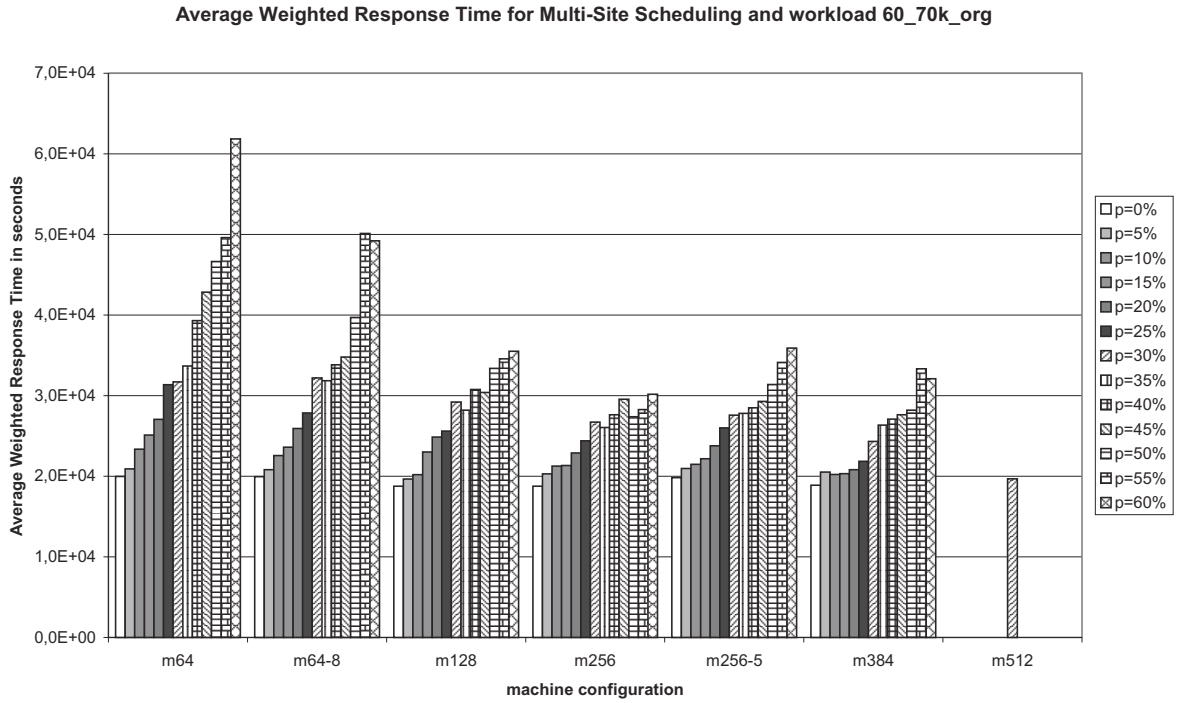

Figure 2 The average weighted response time in seconds for workload 60_70k_org and all machine configurations and Multi-Site Scheduling

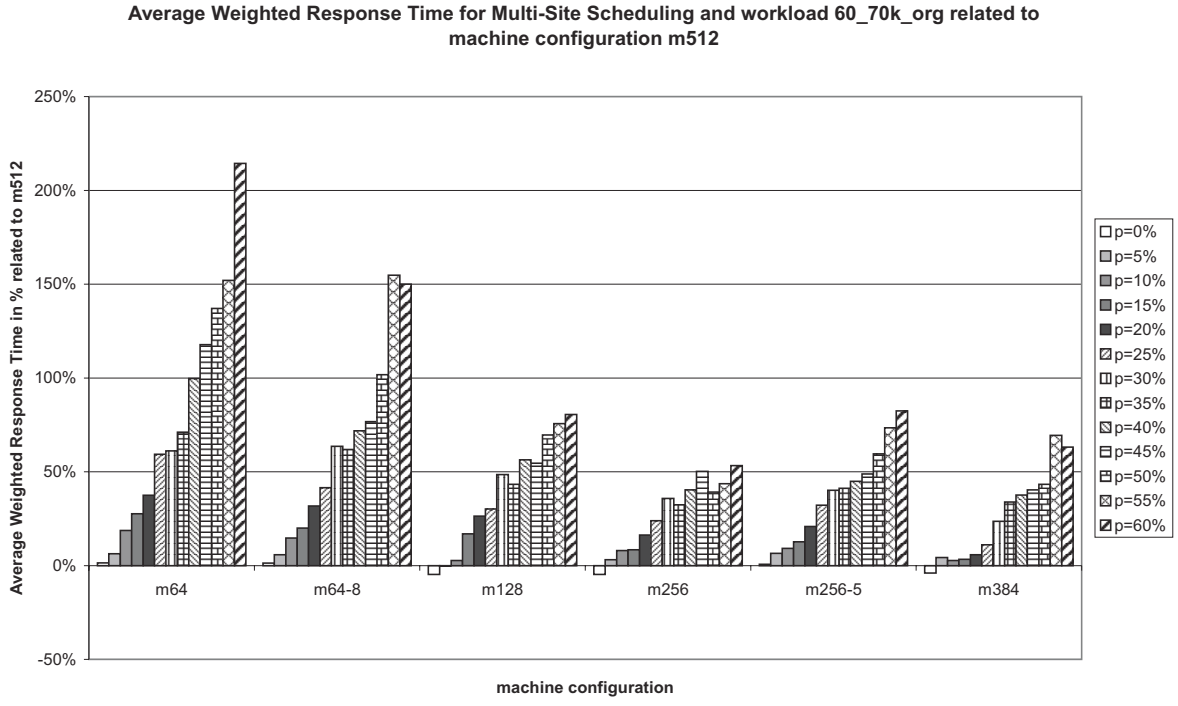

Figure 3 The average weighted response time for workload 60_70k_org and all machine configurations relative to machine configuration m512 for Multi-Site Scheduling

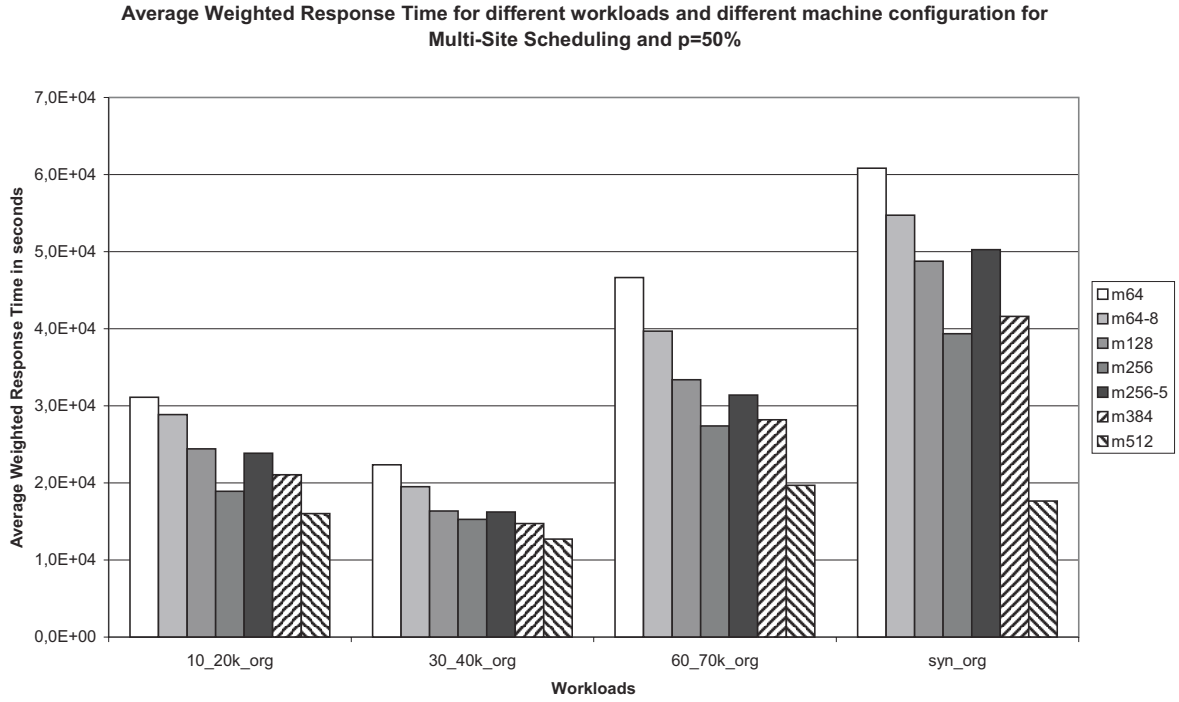

Figure 4 The average weighted response time in seconds for all workloads and all machine configurations for Multi-Site Scheduling and p=50%

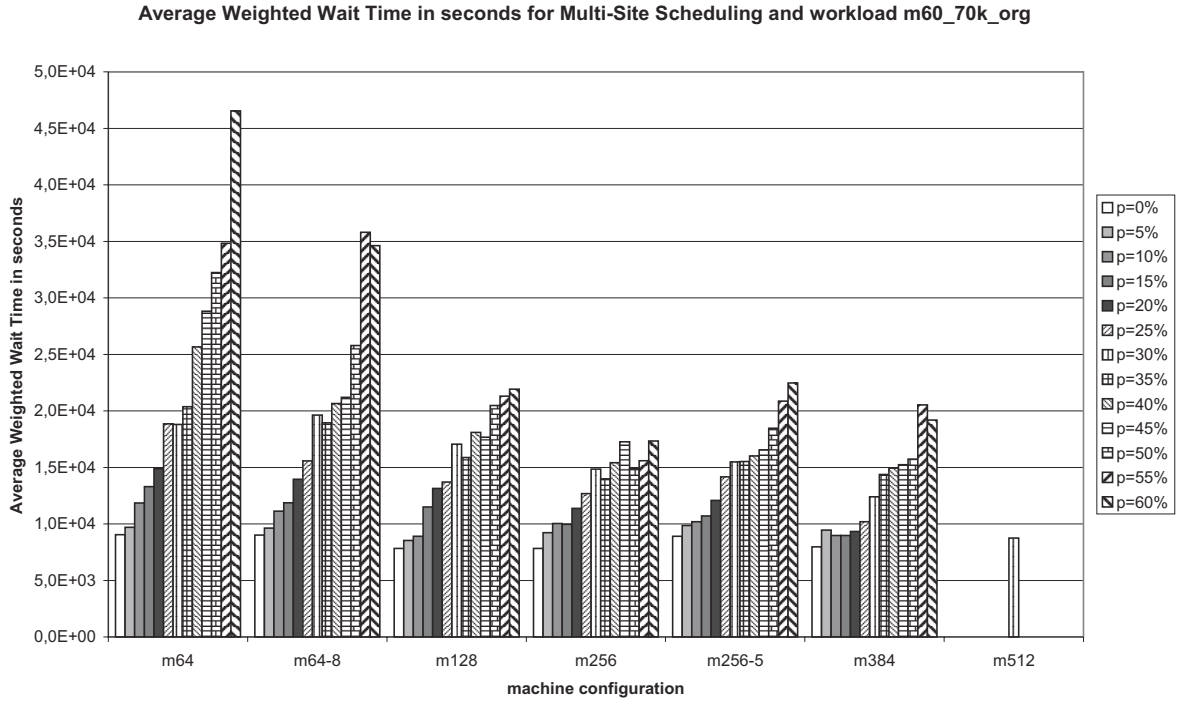

Figure 5 The average weighted wait time in seconds for workload 60_70k_org and all machine configurations for Multi-Site Scheduling

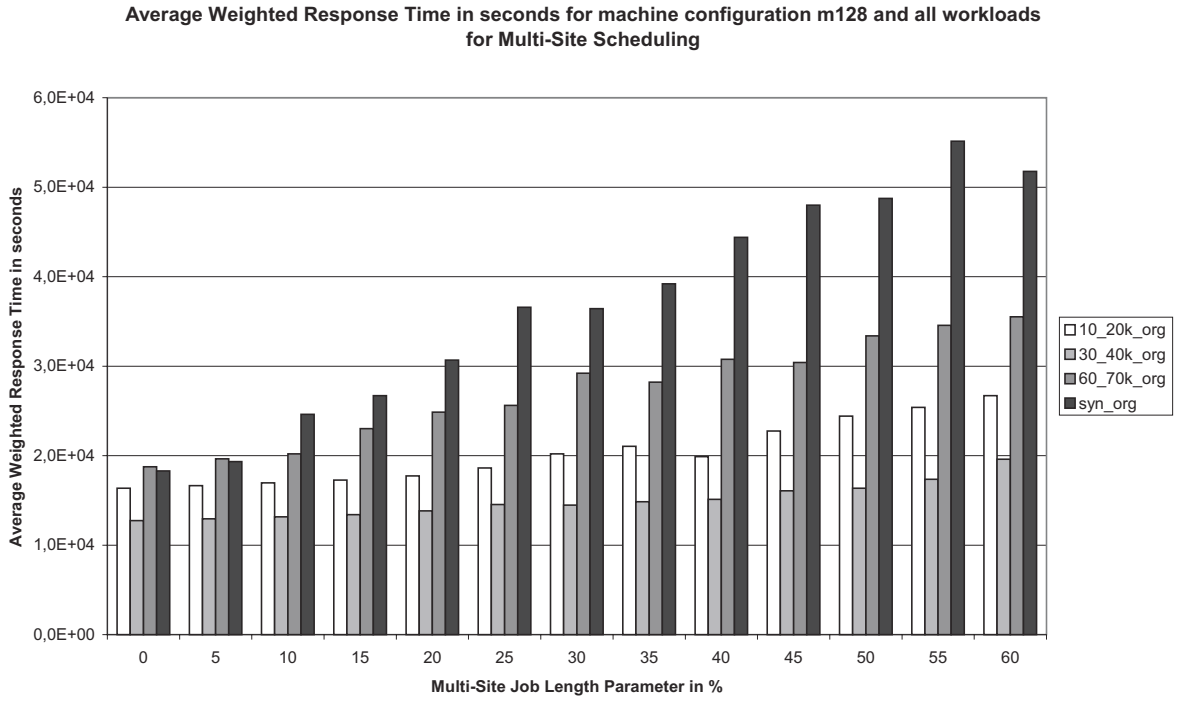

Figure 6 The average weighted response time in seconds for machine configuration m128 and all workloads for Multi-Site Scheduling

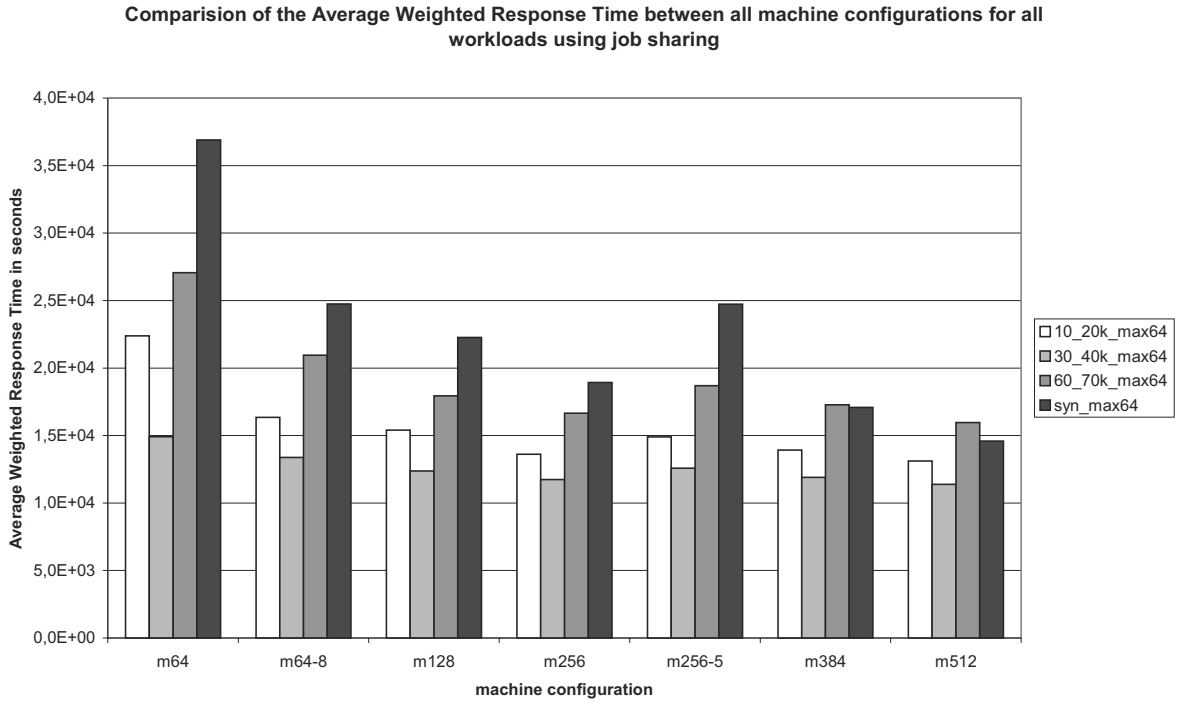

Figure 7 The average weighted response time in seconds for all workloads and all machine configurations for Job Sharing

file	10_20k_org $[jobs] \hat{=} [10^{-2}\%]$	30_40k_org $[jobs] \hat{=} [10^{-2}\%]$	60_70k_org $[jobs] \hat{=} [10^{-2}\%]$	syn_org $[jobs] \hat{=} [10^{-2}\%]$
p=0%	539	431	840	774
p=5%	543	436	850	769
p=10%	582	448	928	827
p=15%	572	490	917	906
p=20%	583	546	946	946
p=25%	601	567	951	1042
p=30%	622	521	979	1026
p=35%	637	523	1036	1063
p=40%	647	534	1106	1012
p=45%	673	597	1008	1181
p=50%	755	579	1029	1188
p=55%	748	578	1086	1233
p=60%	746	638	1114	1177

Table 3 Number of Multi-Site Jobs for different workloads and different parameters using machine configuration m128

file	10_20k_org	30_40k_org	60_70k_org	syn_org
p=0%	22,19%	23,97%	34,01%	40,75%
p=5%	22,87%	25,09%	36,36%	41,21%
p=10%	24,71%	27,21%	37,28%	46,85%
p=15%	25,30%	28,24%	38,97%	47,95%
p=20%	26,05%	31,36%	40,43%	46,91%
p=25%	29,40%	31,93%	41,02%	52,81%
p=30%	29,11%	30,84%	44,53%	53,62%
p=35%	30,24%	31,01%	44,38%	54,01%
p=40%	31,21%	32,82%	48,10%	56,01%
p=45%	33,22%	37,20%	45,87%	59,77%
p=50%	37,15%	34,18%	46,25%	59,99%
p=55%	37,87%	36,05%	49,81%	62,72%
p=60%	41,46%	39,17%	52,38%	60,46%

Table 4 The Squashed Area of the Multi-Site jobs related to the Squashed Area of the whole workload for different workloads and multi-site parameters using machine configuration m128

used job trace. This effect even increases for a higher multi-site parameter p.

As it can be seen in Figure 3 the AWRT for the machine configurations m128, m256 and m384 is smaller in comparison to m512 for the multi-site parameter p=0%. This effect results from the scheduling algorithm for multi-site described in 2.4. Some jobs can surpass queued jobs during the multi-site scheduling, because the scheduler tries to start the jobs as soon as possible or to split them. In contrast, normal backfilling queues the job in first-come-first-serve order.

Figure 4 shows the already mentioned behavior for different workloads while using multi-site scheduling with p=50%. Therefore it is reasonable to assume that the above explained effects apply in general.

The AWWT for simulation with the workload 60_70_org for all used machine configurations is detailed in Figure 5. The value of the AWWT is at least two hours. The average weighted waiting time of about two hours indicates the existence of an appropriate backlog, which is necessary for the backfilling algorithm.

The results presented in Figure 5 show a similar behavior like the results in Figure 2 because the values only differ in the execution time of the jobs.

The next aspect of this study was to identify whether all workloads show the described behavior for each machine configuration. The configuration m128 was chosen as an example to demonstrate the results. All other machine configuration produce similar results.

Figure 6 demonstrates the examined behavior for the AWRT for multi-site scheduling. Depending on the multi-site parameter p the AWRT increases for all workloads. The actual deterioration of performance differs between the workloads.

After the evaluation of multi-site scheduling the results for job sharing will be presented. Jobs with limited resource

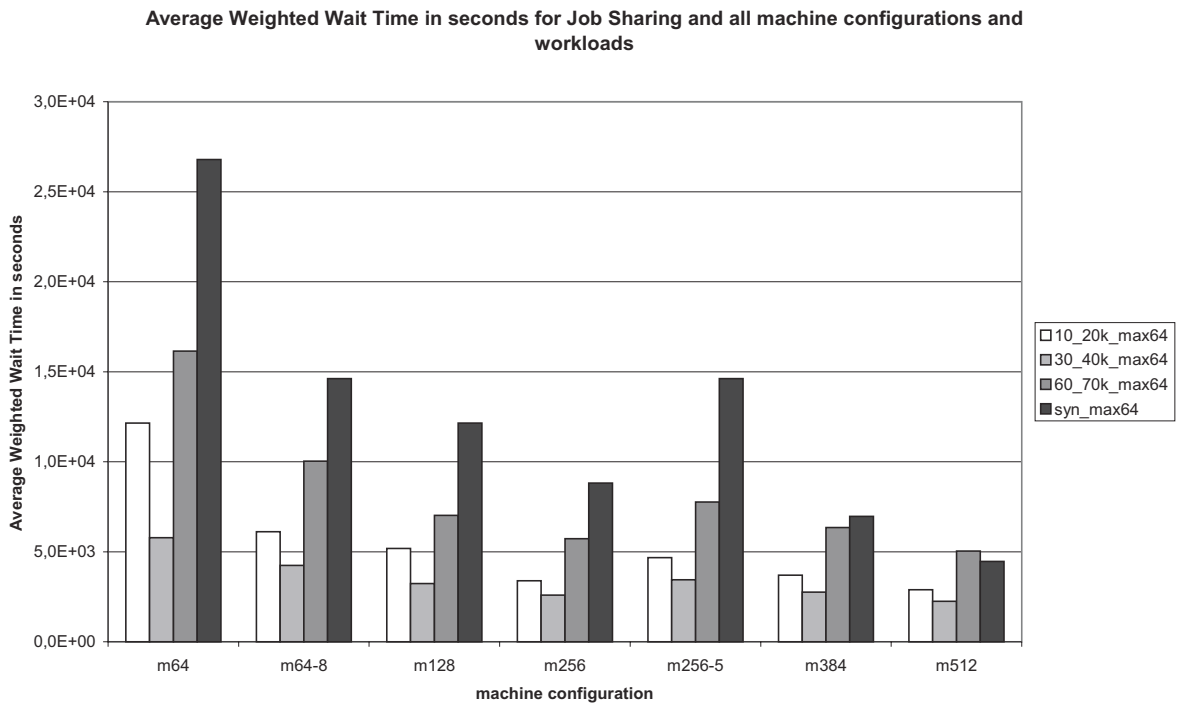

Figure 8 The average weighted wait time in seconds for all workloads and all machine configurations for Job Sharing

demand (e.g. 50% of the largest machines) result only in a minor drawback for configurations with smaller machines. The increase of the average weighted response time stays generally within 20% for all partitioned configurations except m64 compared to a single large machine with 512 nodes. The workload syn_max64 as an exception shows that the results are highly dependent on the workload characteristics.

In Figure 7 the AWRT for all workloads and all machine configurations is presented. The results correspond to the statements made for multi-site scheduling. The AWRT for all workloads and machine configuration m64 is higher than the AWRT for machine configuration m64-8. This indicates that the balanced system with bigger machines is advantageous to the unbalanced system with some smaller machines. The comparison between the configurations m64-8, m128 and m256 shows that the use of bigger machines produces favorably better scheduling results. The analysis of the scheduling behavior of m256 and m256-5 also comes to the same results as for multi-site scheduling. The use of a system with two big machines is preferable to the use of a system with one big and several smaller machines. Again, the comparison between m256 and m384 provides no clear result.

The already mentioned conclusions for the job sharing scenario can be seen in Figure 8 as well. Here the AWWT is presented for all workloads and all machine configurations. The minimal AWWT is about 45 minutes and so there is evidence of a sufficient backlog for the backfilling strategy.

4 Conclusion

Overall, as expected the results show that configurations with large machines are superior to configurations with small machines. However, as long as jobs are limited in resource demand (e.g. 50% of the largest machines) configurations with smaller machines result only in a minor drawback. For example, the increase for the average weighted response time stays in general below 15% for a configuration with four machines of 128 nodes compared to a single large machine with 512 nodes. It can be expected that an adequate ratio between the workload of large jobs and the available computing power of the large machines is necessary to guarantee an acceptable response time. As long as this requirement is met the remaining resources may consist of smaller machines without implying a significant drawback.

In contrast to job-sharing the usage of multi-site scheduling allows the execution of jobs that are not limited by the maximum machine size. As shown in this paper the resource configurations and the overhead due to multi site scheduling have a strong impact on the AWRT of the schedule. Configurations of larger machines are superior to those with smaller machines. Though scenarios with small overheads ($p < 20\%$) only show a slight advantage of balanced large machine configurations compared to unbalanced systems with small machines.

Especially on smaller resource configuration a large overhead results in a very steep increase of the AWRT, as more jobs are executed in multi-site mode in configurations with

a higher number of smaller machines. Nevertheless, there are some factors for the overhead that lead to a decrease or at least no increase of the AWRT compared to smaller factors in the same scenario. This may be caused by two effects. First of all, the number of jobs used for multi site scheduling increases corresponding to the size of the overhead, while the squashed area of these jobs shows a much lower increase than the overhead. Therefore, each job must be smaller in average. This leads to the assumption, that jobs used for multi site scheduling differ in each scenario for each size of the overhead. Second, the increase of the communication overhead leads incidentally to a more suitable job size as certain pattern of job execution times are more common than others.

References

[1] Matthias Brune, Jorn Gehring, Axel Keller, and Alexander Reinefeld. Managing Clusters of geographically distributed High-Performance Computers. *Concurrency - Practice and Experience*, 11(15):887–911, 1999.

[2] European grid forum, http://www.egrid.org, October 2001.

[3] D.G. Feitelson. A Survey of Scheduling in Multiprogrammed Parallel Systems. Research report rc 19790 (87657), IBM T.J. Watson Research Center, Yorktown Heights, NY, February 1995.

[4] D.G. Feitelson and L. Rudolph. Parallel Job Scheduling: Issues and Approaches. In *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing, pages 1-18. Springer Verlag, Lecture Notes in Computer Science LNCS 949, 1995.*, 1995.

[5] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, and K. C. Sevcik. Theory and Practice in Parallel Job Scheduling. *Lecture Notes in Computer Science*, 1291:1–34, 1997.

[6] I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[7] The Grid Forum, http://www.gridforum.org, October 2001.

[8] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. *Lecture Notes in Computer Science*, 1971:191–202, 2000.

[9] S. Hotovy. Workload Evolution on the Cornell Theory Center IBM SP2. In D.G. Feitelson and L. Rudolph, editors, *IPPS'96 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 27–40. Springer–Verlag, Lecture Notes in Computer Science LNCS 1162, 1996.

[10] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the Design and Evaluation of Job Scheduling Algorithms. In *Fifth Annual Workshop on Job Scheduling Strategies for Parallel Processing, IPPS'99; San Juan, Puerto Rico; April 1999*, Lectures Notes in Computer Science, pages 17–42, 1999.

[11] D.A. Lifka. The ANL/IBM SP Scheduling System. In D.G. Feitelson and L. Rudolph, editors, *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer–Verlag, Lecture Notes in Computer Science LNCS 949, 1995.

[12] Miron Livny and Rajesh Raman. High-Throughput Resource Management. In I. Foster and C. Kesselman, editors, *The Grid - Blueprint for a New Computing Infrastructure*, pages 311–337. Morgan Kaufmann, 1999.

[13] U. Schwiegelshohn and R. Yahyapour. Analysis of First-Come-First-Serve Parallel Job Scheduling. In *Proceedings of the 9^{th} SIAM Symposium on Discrete Algorithms*, pages 629–638, January 1998.

[14] U. Schwiegelshohn and R. Yahyapour. Fairness in Parallel Job Scheduling. *Journal of Scheduling, 3(5):297-320. John Wiley*, 2000.

[15] L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.

[16] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs-/parallel/workload/, October 2001.