Job Scheduling for Computational Grids

Carsten Franke*, Uwe Schwiegelshohn, and Ramin Yahyapour

Computer Engineering Institute, University of Dortmund, 44221 Dortmund, Germany,
(email: {carsten.franke,uwe.schwiegelshohn,ramin.yahyapour}@udo.edu)

Abstract. Grid computing is a method to execute computational jobs requiring a significant amount of computing resources and/or large sets of data. Contrary to large heterogeneous distributed systems, a Computational Grid has many independent resource providers with different access policies. In addition to the size of such a Grid, the diversity of those policies leads to a very complex allocation task that cannot be manually handled by the user. This task does not only include the search for suitable resources but also the coordination of the actual job execution on the selected set of resources. Therefore, an efficient and flexible Grid scheduling system is required to manage the job requests of the users. Further, the independent resource providers typically want to maintain control of their Grid resources by use of local management systems. This increases the complexity of the Grid allocation task as those local management systems usually do not provide all system information due to their architecture or due to policy restrictions. Similarly, it is difficult to consider the requirements of individual jobs with complex workflows or complex scheduling objectives. These scheduling objectives are a consequence of the differences in cost and quality of service among the resources. Therefore, Grid scheduling significantly differs from conventional job scheduling on parallel computer systems which has been addressed frequently in the past.
In this paper, we analyze the requirements for scheduling methods in a Grid environment. This leads to implications for the architecture of a Grid scheduling system. Particularly, we discuss several scheduling algorithms with respect to their suitability for Grids. First, we describe how conventional scheduling strategies for parallel systems can be extended to include the Grid paradigm. As resource and job management problems in a Grid can be modelled as a utility market, it seems appropriate to also consider economic and market-oriented methods. Therefore, we discuss those methods and present a market-oriented scheduling strategy and its exemplary implementation. The different strategies are evaluated by use of simulations with trace based workloads for two scenarios. As a result, it is concluded that conventional and economic strategies are both suitable in general for Grid scheduling. Depending on the actual environment constraints in a real Grid system, both strategies have their advantages and their drawbacks which are described in detail. Due to its higher flexibility, the economic approach seems to be the solution of choice for Grid scheduling in the long term.

1 Introduction

The technological progress during the last decades is especially apparent in the vast improvements of computer technology. Although Moore's law has repeatedly been considered to loose validity as physical limits were approached, it still holds very well for many important aspects of computer technology. Nevertheless, computing power remains a limited resource. On the one hand, this is caused by new emerging complex applications which were enabled by the new technology. On the other hand, many existing applications grow in their demand for computing power. Therefore, it is not sufficient to improve the speed of processors, but in addition those processors must be combined to parallel computers in order to satisfy the need for computing power.

Going one step further, combining those parallel architectures with a powerful network naturally leads to concepts for metacomputing or Grid computing which have been subject of many research efforts in the past years. As Grids include resources of different nature as, for instance, CPU, network, data, or software [20, 14], it quickly became a key research goal to offer the user

* born Ernemann.

transparent access to those resources. This transparency is the reason to use the term "Grid" that alludes to the Electrical Power Grid which provides all users with electrical power just on demand without requiring any deeper insight about how and where the power has actually been generated. Similarly, a Computational Grid is supposed to provide computational power on demand to all users without prior knowledge about the locations of the allocated resources. In general, Grids usually denote the sharing of geographically distributed resources that belong to different providers and reside in different administrative domains.

Even without considering the physical locations of the resources and their relations to each other, it would be a cumbersome task for a user to find and utilize all required resources of a job due to the individual access policies of the various providers. Thus, prior to accessing a remote resource, a scheduling and negotiation process is required to automatically allocate computational jobs to available resources. Although several core Grid services are already available, those higher-level services for coordinating the resource access are still missing. Even existing complete Grid systems, like, for instance, Condor, Legion, or Nimrod/G, deliver a full set of services only for specific configurations.

In this paper, the requirements of Grid scheduling are addressed, and appropriate Grid scheduling strategies are evaluated. We consider the scheduling of parallel batch jobs for Grid computing. The workload of a Grid system is generated by independent users who submit their jobs over time. It is the task of the scheduling system to decide when and where a job is executed and to allocate resources to this job. This process is subject to the job requirements, the users' objectives and the resource providers' policies to grant access to the resources. The various steps for performing the scheduling task have been outlined by Schopf [22]. These steps include the finding of suitable resources for such a request, the decision on which actually available resources to chose, and the determination of the start of a particular application. As at least temporarily, more requests are submitted than resources are available, this leads to resource conflicts which must be settled by the scheduling algorithm. This is done under the premise to generate an efficient schedule.

This paper is a summarization of several different approaches for scheduling in Grid environments. It is an extension of our previous work in [11, 9] with a more thorough analysis of the Grid scheduling problem and a comparison of several different scheduling strategies [15]. We first examine extensions of existing conventional scheduling algorithms which are in use on single parallel computers [7] and its extension to multi-site job execution [6] and the achievable benefit for the participation in Grids [10]. In addition, the application of economic, market-oriented methods is discussed and analyzed [11, 9] as this approach provides many practical advantages for Grids. By the very nature of Grid computing in which geographically distributed resources are owned and maintained by different providers who often do not even know each other, the usage of market-oriented models seems appropriate. These models have already been in discussion for computational problems for some time. We evaluate a new economic method by simulations with different workloads that are derived from real traces of parallel computers. In extension to our previous work we provide a more detailed description of the approaches, a comparison and an extended evaluation.

2 Background

While some computational Grids are based on resources within a single administrative domain, like a large company, most computational Grids consist of resources with different providers. In the latter case, most providers are not willing to exclusively assign their resources to the Grid. For instance, a computer may temporarily be removed from a Grid to work solely on a local problem if there is a need for it. In order to react immediately in those situations, providers typically insist on local control over their resources which is achieved by use of a local management system.

On the other hand, as already mentioned, it would be a cumbersome and tedious task for a potential Grid user to manually select all resources needed to run his Grid application. To prevent those tasks from significantly slowing down the proliferation of computational Grids, a specific Grid management system is needed. Ideally, such a Grid management system includes a

separate scheduling layer that collects the Grid resources specified in a job request, considers all requirements, and interacts with the local scheduling systems of the individual Grid resources. Hence, the scheduling paradigm of a Grid management system will significantly deviate from that of local or centralized schedulers for large computer systems which typically have immediate access to all system information. Although it seems obvious that a Grid scheduler will consist of more than a single scheduling layer, the details of an appropriate scheduling architecture have not yet been established [25]. Nevertheless, it is clear that some layers are closer to the user (**higher-level scheduling instance**) while others are directly affiliated with the resource (**lower-level scheduling instance**). Of course, those different layers must exchange information among each other.

In general, Grids may not be restricted to two levels of scheduling layers. For instance, a large computational Grid may consist of several Sub-Grids. Each of those Sub-Grids has a separate Grid management system. The Grid scheduler of one Sub-Grid may decide to forward the request of a local user to another Grid scheduler from a second Sub-Grid. Then, this second Grid scheduler interacts with the local scheduling systems of the resources in its Sub-Grid. Clearly, three scheduling instances are involved in this situation. The Grid scheduler of the second Sub-Grid is a lower-level scheduling instance with respect to the Grid scheduler of the first Sub-Grid, while it is a higher-level instance in the communication process with the local scheduling systems of the second Sub-Grid. Of course, the scheduling instance in the lowest layer is always a local scheduling system.

Presently, a variety of different local scheduling systems, like PBS, LoadLeveler, LSF, or Condor [19], are installed in large computer systems. A Grid scheduling layer must exploit the capabilities of those local scheduling systems to make efficient use of the corresponding Grid resources. However, those capabilities are not the same for all local scheduling systems due to system heterogeneity.

All this leads to several general aspects that must be considered in designing a generic Grid scheduling environment, see also Czajkowski et al.[5]:

Site Autonomy All resources of a Grid are typically not owned and maintained by the same administrative instance.

Independent Schedulers A Grid scheduler has no exclusive control over all resources in a Grid. Therefore, it must cooperate with the existing independent local schedulers.

Heterogeneous Substrate The resources usually have their own local management software with different features depending on the local policies and functionalities of the management software. Hence, the Grid scheduler has to cope with the limitations of the local management.

Online Problem The Grid scheduling takes place in an online environment where jobs are submitted at any time. Additionally, information on the current state of resources may be difficult to obtain and to keep up to date.

Scalability and Reliability As the Grid is intended to span over a very large number of systems, a Grid management system requires a high degree of scalability. The general Grid management must remain operable even if some components are impacted by a resource failure.

Variable Scheduling Objectives The scheduling objectives may vary for each available resource according to the provider's policy. In addition to the objectives of the provider the needs of the users must also be taken into account.

Co-allocation Some applications need several resources from different providers at the same time. The Grid scheduling system must be capable of coordinating these resources.

Resource Reservation Several complex applications require the reservation of resources in advance. Further, it is advantageous for the scheduler to consider system downtime or restricted access that is known beforehand. Finally, advance reservations are required for co-allocated resources or multi-site applications.

For the design and evaluation of Grid scheduling strategies, we distinguish two Grid computing scenarios.

1. **HPC Grids:** Cooperation between a limited or moderate number of computing sites with high performance computer systems and a dedicated user community.

2. **Global Grid:** Large-scale Grids with a diverse number of resources and independent users.

The first scenario is currently the typical use case of a Grid. The relatively small user community is part of a scientific community and often originates at one of the participating sites. This scenario is also applicable for large commercial companies in which existing computing resources, for instance at different locations, are used efficiently.

The second scenario represents the idea of a global large-scale Grid infrastructure in which individual users have access to a large number of Grid resources. These resources belong to independent and typically unknown providers. This scenario is the generally discussed and anticipated long-term vision for Grid technology.

In the following, we analyze these two Grid scenarios. Both have different requirements which must be taken into account when designing a Grid scheduling system. For instance, the scalability issue is less important in the first scenario. There, it may not be necessary to pay much attention to complex scheduling objectives including cost management and commercial Grid business models. Therefore, we may use a different approach on Grid scheduling in this scenario by extending existing scheduling methods for high performance computer systems. For evaluation purposes, however, we use the same generic Grid model for both scenarios which allows us to compare the presented methods later. But note that especially the second scenario covers a larger diversity of Grids.

Our computational Grids consist of several independent computing resources which are linked by interconnection networks. The term *independent* emphasizes the fact that the participating resources are not controlled by a single entity and may be geographically distributed. While this definition for a Grid does not determines details of the individual resources in the network, in practice, these resources are often high performance computing components, like massive parallel processors, networks with high speed and high bandwidth, or large databases. Only the component at the user access point may not necessarily belong to this category. However, the user of such a Grid system may not be aware of the internal system structure but has the illusion of a single virtual machine. To support this concept, the Grid management system may use several components of the system concurrently to solve a large problem. This is called multi-site execution.

In our studies, we assume that each participating site has a single massively parallel machine that consists of several nodes. A parallel job can be allocated to any subset of nodes of a machine. This model comes reasonably close to a Grid environment consisting of real systems like an IBM RS/6000 Scalable Parallel Computer, a Sun Enterprise 12000, or an HPC cluster.

For simplicity all machines and 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 additionally limit the number of suitable machines for a job. In a real implementation, a preselection step is part of the Grid scheduling process and is normally executed before the actual scheduling takes place. After this preselection phase, the scheduler ideally chooses from several resources that are all suitable for the job request. In this study, we neglect this preselection step and focus on the remaining scheduling task. Therefore, in the following, it is assumed that all job parts could be executed on any node.

The jobs are not preempted nor time-sharing is used. Thus, once started, a job runs until completion. Furthermore, we do not consider the case that a job exceeds its allocated time. After its 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 neither moldable nor malleable [13, 12].

As Grid workloads are not yet available, we model a Grid scenario with the help workloads for single parallel machines. Here, jobs are submitted by independent users at the local sites. This produces an incoming stream of jobs over time. Thus, we deal with an on-line scheduling problem without any knowledge on future job submissions. The scheduling system allocates resources for the jobs and determines its starting time. Then, the job is executed without any further user interaction.

In a real implementation, the job data must be transferred to the remote site before the execution. This transport of data requires additional time. This effect can often be hidden by

Grid data management services which pre-fetch data before the execution and return output data afterwards. In this case the resulting overhead is not necessarily part of the scheduling process. For now, we neglect this data transport for the scheduling and discuss this decision in the later evaluation.

3 Evaluation of Load-Sharing in Grids

Within this section, we address Grid scheduling strategies for the above mentioned Scenario 1. As already pointed out the different scheduling strategies are derived from conventional scheduling on parallel computers. In this first study, we consider different levels of cooperation methods between the computing sites and analyze the impact on the scheduling quality in terms of minimization of response times The scheduling algorithms are mainly based on the well known first-come-first-serve (FCFS) strategy and its derivatives [18].
We start by considering the following 3 different cases.

1. **Local Job Processing**

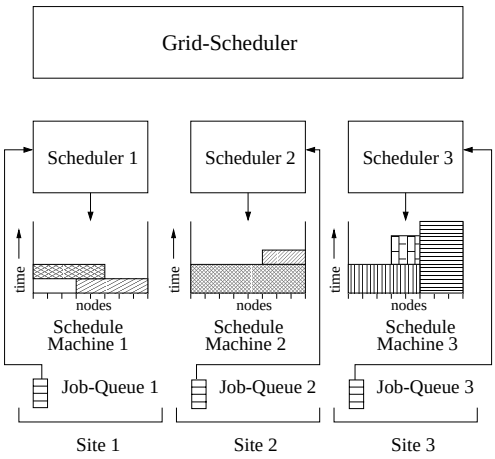

Fig. 1. Sites Executing All Jobs Locally.

Here, the computing resources at a site are dedicated only to their local users (see Figure 1). Hence, each site has its own workload that is not shared with other sites. For our simulations, we use the EASY backfilling algorithm of Lifka [18] that has been implemented for several installations of high performance computers [26].

2. **Job Sharing**
In this case, all jobs are forwarded to a central Grid scheduler as seen in Figure 2. Note that this scheduler may be implemented in a distributed fashion. Such an implementation requires an additional management overhead for exchanging information about the current state of the queues and schedules in the system. Nevertheless, it knows about all submitted jobs. This is only possible in the HPC Grid Scenario 1 where the number of different sites is limited. However, in practice a local system may decide to forward only a subset of its local job to a Grid scheduler. The scheduling algorithms consist of two steps representing two layers of a Grid scheduler. In the first step, the machine is selected, while in the second step, the actual scheduling on the target machine is executed.
Step 1 - Machine Selection: There are several methods possible for selecting the target machine. For instance, other simulations [15] showed good results for a selection strategy called *BestFit*. Here, the machine is selected on which the job would leave the least number of idle resources if it is started there as soon as possible.

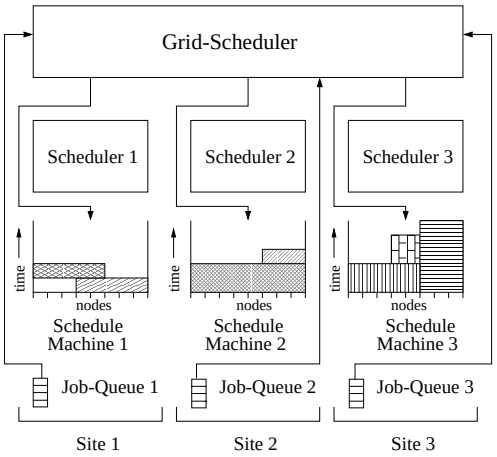

Fig. 2. Sites Sharing Jobs and Resources.

Step 2 - Scheduling Algorithm: Again, EASY backfill is used.

3. **Multi-Site Computing**
   This case allows the concurrent execution of a job on more than one machine. It comes closest to the concept of a single large virtual machine while the usage of multi-site applications has been theoretically discussed for quite some time [2]. There are only few real multi-site applications in practice. Therefore, there are no data to determine the effect of multi-site execution on the execution time of a job.
   In our model, the local schedulers forward all jobs to a Grid scheduler. This time, the Grid scheduler does not simply select a single site for job execution but may also split jobs to be executed across site boundaries (see Figure 3). Note, that the job parts still run in parallel on the different sites.

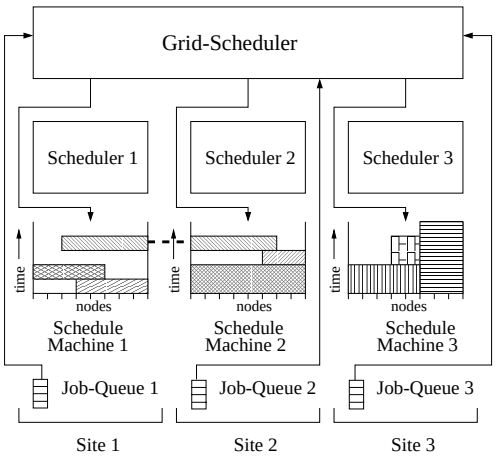

Fig. 3. Support for Multi-Site Execution of Jobs.

Again, there are several strategies possible for multi-site scheduling. Here, we use a scheduler that first tries to find a site with sufficient currently idle resources for immediately starting the job. If such a machine is not available, the scheduler tries to allocate resources from different sites for the job. To this end, the sites are sorted in the descending order of idle resources

which are then allocated in this order to the job. This way, the number of execution sites is minimized. If there are not enough free resources available for a job, it is queued and backfilling is later applied.

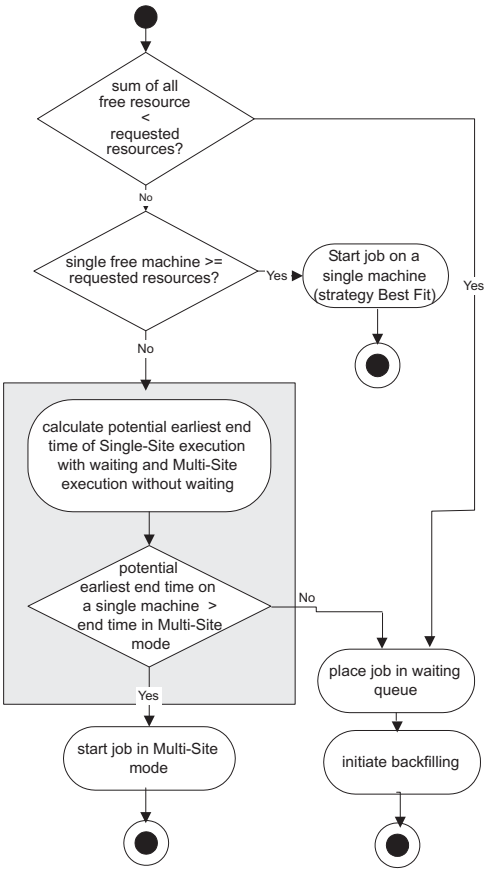

Fig. 4. Algorithm for Multi-Site Scheduling.

Spreading job parts over different sites usually produces an additional overhead. This overhead is a result of the process communication during run-time which must be partially executed over the WAN. However, as WAN networks become faster, this overhead may decrease over time. Further, it depends on the particular application. For those jobs with limited communication demand there is only a small impact. The overhead in a real Grid scenario highly depends on the job communication pattern and the network configuration between the sites. Due to the lack of available data, we assume a proportional increase for the execution time of all multi-site jobs. To this end, we model the influence of the overhead by extending the required execution time r_i to r_i^* for a job i that runs on multiple sites by a constant factor:
$r_i^* = (1 + p) \cdot r_i$ with $p = 0$ .. 300 % in steps of 5%.
Note, without the introduction of any penalty for multi-site execution, the Grid would behave like a single large computer. Hence, multi-site scheduling will ideally outperform all other scheduling strategies.
Further, we also examined an improved multi-site strategy. In this algorithm, the earliest potential completion time of the job running on a single machine is compared to the completion time of a multi-site job execution with the additional overhead. The better alternative is chosen, see Figure 4. Note that this is only an estimation as the real starting time of the job may vary depending on the completion time of jobs ahead in the queue.

3.1 Machine Configurations

We considered several Grid configurations in our evaluation to determine the robustness of our results. In order to allow comparisons between the results of different configurations, we use the same workloads and require that the sum of all nodes in each configuration comprises exactly 512. Those nodes are partitioned into various machine configurations as shown in Table 1.

The configurations *m64-8*, *m128-4* and *m256-2* represent several sites with 8, 4 and 2 identical machines, respectively. They are balanced as there is an equal number of nodes at each machine. The configurations *m384-6* and *m256-5* are examples of a large computing center with several smaller client sites.

Finally, the reference configuration *m512-1* consists of a single site with one large machine. In this case no Grid computing is used. This reference configuration yields the best scheduling output which is possible without partitioning the compute resources into several machines and without any overhead due to Grid scheduling.

identifier	configuration	max. size	sum
m64-18	$4 \cdot 64 + 6 \cdot 32 + 8 \cdot 8$	64	512
m64-8	$8 \cdot 64$	64	512
m128-4	$4 \cdot 128$	128	512
m256-2	$2 \cdot 256$	256	512
m256-5	$1 \cdot 256 + 4 \cdot 64$	256	512
m384-6	$1 \cdot 384 + 1 \cdot 64 + 4 \cdot 16$	384	512
m512-1	$1 \cdot 512$	512	512

Table 1. Resource Configurations.

3.2 Workload Model

The evaluation of Grid scheduling strategies is not only based on the chosen machine configuration but also on the workload models. In the following, we present the workload models which we used in the simulations of our evaluation.

It is known that scheduling systems are sensitive to the workload. Therefore, it is important to select realistic workloads. They can either be based on workload models or on real workloads. In the first case, a job generator produces a stream of jobs based in the workload model. Significant research efforts have been done for creating appropriate models [28, 27, 21]. However, those models are still not able to express all relevant characteristics of real workloads. For instance, many hidden characteristics like temporal patterns, dependencies between jobs, or user behaviors are rarely considered.

Due to this lack of appropriate workload models, we derive suitable input data from real job traces which are available in the Standard Workload Archive [29]. In order to use these traces for our study, it is necessary to modify them to simulate submissions at independent sites with local users. To this end, the jobs from the real traces are assigned in a round-robin fashion to the different sites. However, it is typical for many known workloads to favor jobs requiring 2^x nodes. For example, the used workload from the Cornell Theory Center (CTC) shows this characteristic too[16]. Configurations with smaller machines would be put into disadvantage if the number of nodes on these machines is not a power of 2. To this end, we selected configurations with a total number of 512 nodes.

As we already discussed, the quality of a scheduler highly depends on the used workload. To minimize the risk that singular and abnormal workload characteristics affect the validity of our results, the evaluation simulations have been done for 4 different workload sets which are all taken or derived from the same user group of the corresponding real installation.
Specifically, we use:

- three extracts of original CTC traces and
- one synthetic workload generated by a probabilistic workload model on the basis of the CTC traces.

Each workload set consists of 10000 jobs which corresponds in real time to a period of more than three months.

The synthetic workload is very similar to the original CTC data set. It has been generated to prevent that potential singular effects in real traces, like a down-time of the real system, affect the accuracy of the results.

Special care has to be taken in our simulation for handling the "wide" jobs which are contained in the original workload traces. These are jobs within the workload which require more processing nodes than available on a machine. For instance, the widest job in the CTC traces requests 336 processing nodes. It cannot be executed in all of our Grid configurations. 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. The wide jobs are split up into several parts of the local machine size to allow their execution (*Modification 1*). We also examined two other workload modifications. In one case, the job size is limited by the size of

the largest machine in the configuration (*Modification 2*). Here, users can submit jobs that are wider than the local machine size. In another case, all jobs are split up into several parts with maximum 64 nodes to allow their execution on all examined configurations (*Modification 3*). Every configuration has a machine that consists of at least 64 nodes. To allow the comparison of different configurations for job sharing the following modifications have been applied to each aforementioned workload.

The workloads with modification *1* and *3* were executed in all 3 scheduling cases. The workloads with modification *2* were simulated for the *job-sharing* and *multi-site* case. Note, that all of these modifications do not alter the overall amount of workload. Wit our modifications large jobs are split up, but they are still generated at the same site. Depending on the case, a user may submit jobs larger than locally available. The simulations allow the examination of the impact caused by wider jobs on the schedule.

identifier	description
W1	An extract of the original CTC traces from job 10000 to 20000.
W2	An extract of the original CTC traces from job 30000 to 40000.
W3	An extract of the original CTC traces from job 60000 to 70000.
W4	The synthetically generated workload derived from the CTC workload traces.

Table 2. The Used Workloads.

The input workloads are summarized in Table 2 and an identifier is introduced for each workload. The selection of workloads and the application to different machine configurations prevents the over- or underutilization of machines. If there are not enough jobs to be executed on a machine, the results may lead to wrong interpretations due to unrealistic modelling. The backlog of a scheduling system is a good indicator to check the utilization. The backlog is the workload that is queued at any time instant as there are not enough free resources to start the jobs. A small or even no backlog indicates that the system is not fully utilized. In our evaluations, we could show that all traces provide enough workload to keep a sufficient backlog on all systems (see also results from [15, 8]): The average weighted wait time of all jobs is between 45 minutes and more than two hours which indicates the existence of an appropriate backlog. The examination of the actual schedules also shows that there was no job starvation. That is, the backlog did not grow significantly during the simulation which would have been the case if the machines are not capable to master the workload.

3.3 Simulation Results

For the evaluation of the different structures and algorithms discrete event simulations have been performed, see also [8, 7, 6]. We use the average weighted response time (AWRT) as the measure in this study. Note that each job j in the set of all jobs τ is released at time r_j. The response time of each job j is weighted by its processor number m_j multiplied by its runtime p_j. Furthermore, the completion time of job j within the resulting schedule S is denoted by $C_j(S)$. This weighting prevents any prioritization of small over wider jobs in terms of the average weighted response time if no resources are left idle [23, 24]. Thus, the average weighted response time is defined as follows:

$$\text{AWRT} = \frac{\sum_{j \in \tau} (m_j \cdot p_j \cdot (C_j(S) - r_j))}{\sum_{j \in \tau} m_j \cdot p_j}$$

Job-Sharing The usage of job-sharing improved the average weighted response time in all examined machine configuration and in all workloads in comparison to the local job execution. In the *m128-4* configuration for example the improvement is over 50% (see Figure 5). The achieved

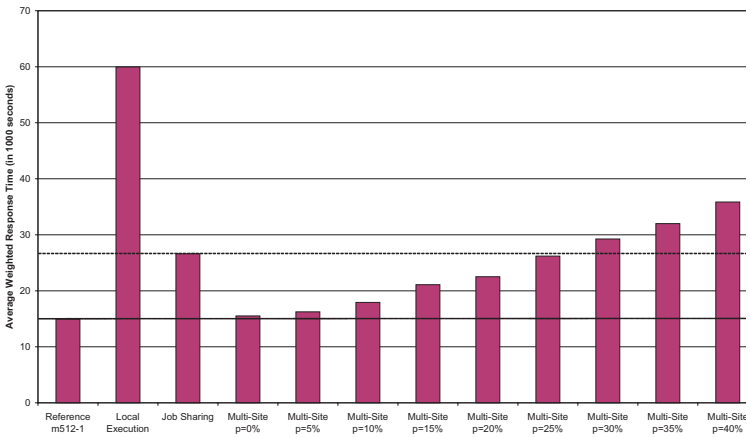

Fig. 5. Average Weighted Response Time for the *m128-4* Configuration and Workload *W4* with Modification *2*.

results are similar for all other simulations. As mentioned before large jobs that are wider than the local machine have been split up into smaller jobs which are sequentially executed on the local system. This leads to an increase of the AWRT as the workload of these wide jobs is still generated and submitted at the same time as the original wide job, but the sequential execution leads to a delay of job parts. Job-sharing on the other hand allows the transfer of jobs to remote machines.

Multi-Site Computing Next, we examine the influence of using multi-site execution. The results show that further improvements on the AWRT can be achieved in comparison to job-sharing. As a reference the result for a single machine with 512 nodes is used (Figure 5). As expected, the average weighted response time without overhead for multi-site is near to the *m512-1* result, see Figure 5. The difference is caused as the multi-site strategy is quite simple and does not fully exploit the potential of all available machines.
Moreover, multi-site execution is also beneficial compared to job-sharing even for an overhead of about 25% on the execution time (Figure 5). Similar results are achieved for other configurations, see [7, 8]. The mentioned results, as given in Figure 5, showed the least effective improvements in comparison to other examined workloads. The average weighted response time in other configurations delivered even better results. Therefore, the tolerable overhead on multi-site executed jobs can even be larger.

Examining the sensitivity on different machine configurations, the simulation results show that configurations with larger machines provide significant better scheduling results than machine configurations that are higher fragmented, see Figure 6 for all machine configurations and workload W4.

The increase of the AWRT results from two effects. First, the overall workload increases as all multi-site jobs have a longer execution time due to the overhead (p). Second, we can make the observation that the number of multi-site jobs increases with additional overhead. Table 3 shows the number of multi-site jobs for machine configuration m128-4 for different workloads and different multi-site parameters p. The effect can be seen for all workloads. This process is not monotone, 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 idle times within the schedule decreases and the probability of free resources within one machine decreases as well. Therefore more jobs are started in multi-site mode.

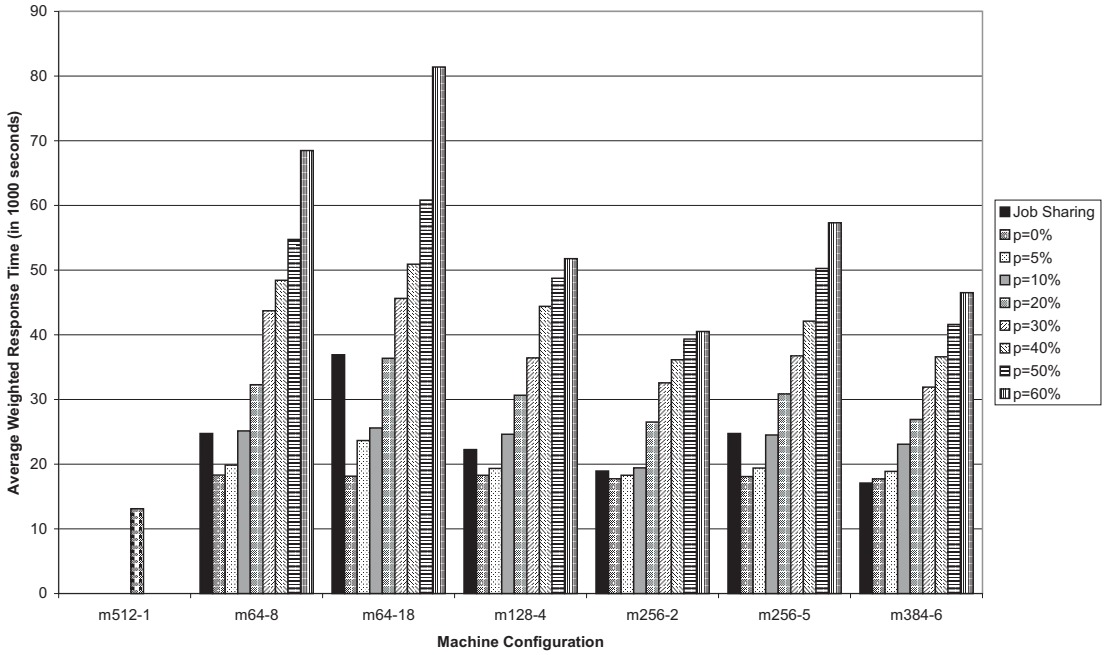

Fig. 6. The Average Weighted Response Time in Seconds for Workload *W4* and all Machine Configurations and Multi-Site Scheduling.

The previous results showed that jobs running in multi-site mode are in majority jobs with a higher resource demand [8, 7]. This can be concluded because 5% to 10% of all jobs are multi-site jobs while they are responsible for about 20% to 40% of the whole amount of workload depending on the used job trace. This effect even increases for a higher multi-site overhead (p) and is responsible for the mentioned results. However, for machine configurations m384-6 job sharing always yields better results than multi-site scheduling. This effect results from the algorithm for multi-site scheduling. Here, a job that is considered for multi-site execution can be started earlier as previously submitted but not yet started jobs. In that case, this strategy does not consider backfilling for the waiting jobs.

The job sharing and multi-site execution of jobs in configurations with bigger machines is advantageous to configurations with more smaller machines. In our example the m64-8 configurations produces to upto 25% better AWRT results in comparison to m64-18. The comparison between the configurations m64-8, m128-4 and m256-2 shows that the use of bigger machines produces favorably better scheduling results.

Figure 7 shows the average weighted response time for the improved multi-site scheduling. The improvements can be seen in comparison to Figure 6. In general, the average weighted response time is lowered for each configuration by the application of this improved strategy. That is, the Grid in these settings can handle multi-site jobs with a higher penalty while still improving against the job sharing scheduling case. However, as it can be seen in the Figure, for configurations with large parallel machines job sharing often still yields better results than multi-site computing.

Note, that in resource configuration *m384-6* multi-site execution is not necessary, as the largest job within the workload requests only 336 nodes and can therefore be executed on the machine with 384 nodes. Even an overhead of 300% in resource configuration *m384-6* yields 4% execution of all jobs in multi-site mode. This is only a 42% reduction compared to an overhead of 30% in the same configuration. Whereas the resource consumption of the multi-site jobs decreases to 30% in the same case. Overall, increasing the overhead ten times from 30% to 300% only results in an increase of the average weighted response time of about 7% in this configuration.

file	W1 $[jobs] \hat{=} [10^{-2}\%]$	W2 $[jobs] \hat{=} [10^{-2}\%]$	W3 $[jobs] \hat{=} [10^{-2}\%]$	W4 $[jobs] \hat{=} [10^{-2}\%]$
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 *m128-4*.

Note, we do not conclude that multi-site is suitable for all applications. WAN networks are in terms of latency in the order of 2-3 magnitudes slower than common fast interconnection networks between nodes inside a parallel computer, e.g. an IBM SP Switch. Thus, the actual overhead caused by multi-site may be much higher. The question if multi-site execution is suitable, depends on many factors as e.g. the actual communication pattern, the requirement in data I/O of input/output data. Nevertheless, the results indicate that multi-site execution may be beneficial in terms of response time reduction for applications with a limited demand in communication as presented in our results.

Our evaluations showed good results for the average weighted response time. Other examinations not presented here showed also that high utilization of the machines were achieved.

4 Market-Oriented Scheduling

In the following we examine the Global Grid scenario in more detail. In such a Grid environment, there are much more different users and providers which typically do not know each other. Here, the complete scheduling considerations presented in Section 2 is taken into account. While the optimization goal on a single parallel machine or in a small Grid is usually the minimization of the completion time of a computational job, in a large-scale Grid we might encounter more complex scheduling objectives. Here, criteria have to be considered like for instance cost, quality of service or additional time constraints, e.g. given start and completion time limits. The examined algorithms in the previous sections are mostly derivations of list-schedulers that are usually predestined for a single overall objective.

For the Global Grid scenario, other scheduling approaches are necessary that may deal better with different user objectives as well as provider and resource policies. The independent users or resource providers in future Grids can be compared to a society in which independent parties trade for goods; in our case a good is the node allocation for a certain time. This is a typical application scenario in which economic models are used. In the following, we want to address the idea of applying economic models to the scheduling task. To this end a market-economic method for Grid scheduling is presented and analyzed [9]. In our study we use the same evaluation process from the previous sections. However, the quality of an economic scheduling process is more difficult to measure as the former single objective is now dependent on the specific and variable objective formulation and the corresponding cost-metric. While there is a high degree of freedom in these processes, heuristics are often applied to limit the necessary time to exert the market methods. Because economic methods support different objectives, it is difficult to compare and

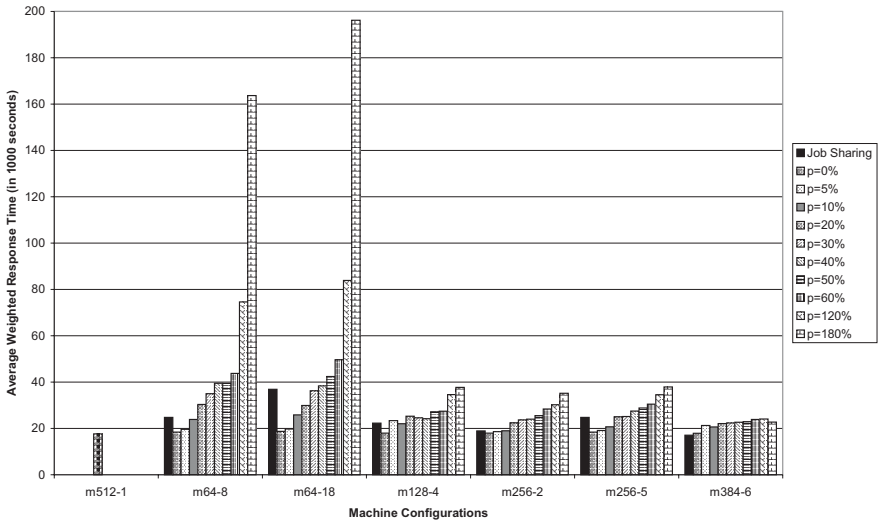

Fig. 7. Comparison of Adaptive Schedules for Workload *W4*.

evaluate such methods to other approaches. As there are no information about Grid workloads nor user objectives and cost models available, we compare the market-oriented with the conventional scheduling strategies as presented in Section 3. Therefore, we consider again only the response time minimization in our simulations. However, here we do not exploit all additional features of the economic model like the support for variable objective formulation.

4.1 Market Methods

Market methods for scheduling computational power have been subject of research for quite some time. Such economic methods have also been applied in various contexts including Grids, see the references given by Buyya [4] or by Cheliothis and Kenyon [17]. The supply and demand mechanisms provide the possibility to optimize different objectives of the market participants. It is expected that such methods provide high robustness and flexibility in case of failures and a high adaptability during changes.

It has to be emphasized that a market method is an equilibrium protocol and not a complete algorithm. Various methods, like the execution of *auctions* [33], have been developed to obtain this equilibrium. Common examples for auctions are: the English Auction, the Dutch Auction, the Sealed Bid Auction and the Double Auction, see Walsh et al. [31]. More details about the general equilibrium and its existence are given by Ygge [34].

In the following, we present our economic scheduling method for Grids. In contrast to the above mentioned approaches, we consider the flexible definition of scheduling objectives for each individual job request and each resource offer. Earlier applications of similar methods can be found in the *WALRAS* and *Enterprise* method. As our strategy is derived from these methods, we briefly introduce their main concepts.

The *WALRAS* method is a classic approach that translates a complex, distributed problem into an equilibrium problem [1]. It is based on the assumption of *perfect competition*, that is, the agents representing the market participants do not try to manipulate the prices with the help of speculation. To solve the equilibrium problem, the *WALRAS* method uses a *Double Auction*. During that process, all agents send their utility functions to a central auctioneer who calculates the equilibrium prices. A separate auction is started for every good. At the end, the resulting prices are transmitted back to all agents. As the utility of goods may not be independent for the agents, they can react on the new equilibrium prices by re-adjusting their utility functions.

Subsequently, the process starts again. This iteration is repeated until the equilibrium prices are stabilized. As we will show later, we substitute in our application the central auctioneer by a decentralized equilibration algorithm.

The *Enterprise* [30] system is another example for market methods. Here, machines create offers for jobs to be run on those machines. To this end, all jobs describe their necessary environment in detail. After all machines have created their offers, the jobs select between these offers. The machine that provides the shortest response time has the highest priority and will be chosen by the job. All machines have a priority scheme where jobs with a shorter run time have a higher priority. Although our model is based on this Enterprise method, it is not restricted to a single objective function.

4.2 Economic Scheduling Model

In contrast to Buyya et al. [3, 4] or Waldspurger [32], our scheduling model does not rely on a single central scheduling instance. Moreover, we use a peer-to-peer like approach in which all participants act independently and may have different objectives and policies. Our method also supports the co-allocation of distributed resources in different domains. This feature is useful for multi-site job execution.

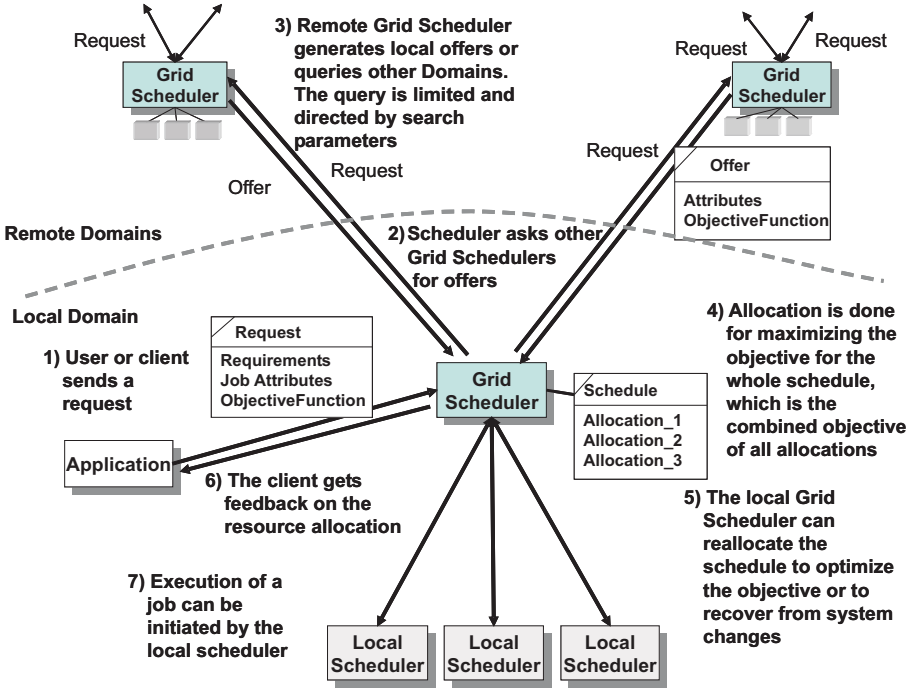

Fig. 8. Scheduling Steps.

The general application flow in our grid job scheduling infrastructure is presented in Figure 8. We assume that each user has access to at least one computing site which may be his local resource provider. In our scheduling model a user submits his job request to the scheduler of his local domain. The scheduler first analyzes this request and, if possible, creates new offers for all local machines. After this step, a first selection takes place in which only the best offers are selected. The remaining request is forwarded to the schedulers of other known domains. This is possible as long as the search depth of involved domains for this request is not exceeded and the *time to live*

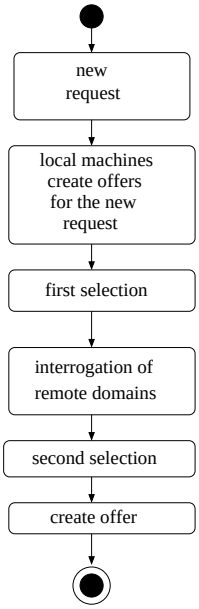

Fig. 9. General Application Flow.

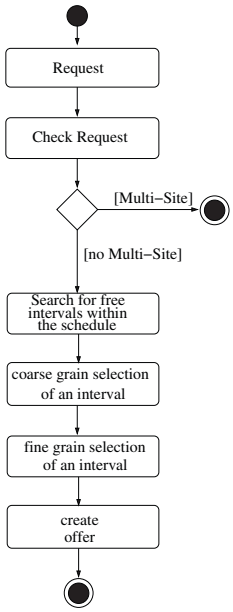

Fig. 10. Local Offer Creation.

value for this request is still valid. Several strategies are possible to select which domain is queried next for offers.

This peer-to-peer approach allows the exploitation of network-locality as users can query their local or nearest domain first and the request is then forwarded to related sites. To this end, each domain manages a list of other domains it may query for offers. Such a list can be manually or automatically maintained with the option to consider a network structure or logical hierarchy. Furthermore, it is possible to include information services that provide addresses for domains and information on the available resources. Note that the presented model can be applied to Grid infrastructures with dedicated information services as well as to completely decentralized peer-to-peer configurations.

The remote domains create new offers and return their best offer to the requesting domain. A domain does not answer to a request for a job if this request has already been processed before, that is, a domain answers any request for a job only once in case the same request may be forwarded from different domains. Afterwards, a second selection process takes place in order to find the best offers among the returned results of this particular domain.

4.3 Economic Scheduling Algorithm

This section includes a description of the scheduling algorithm that has been implemented for the presented infrastructure. The general application flow can be seen in Figure 9.

Note that this method is an auction with neither a central nor a de-central auctioneer. Moreover, the different objective functions of all participants are used for the equilibration process. For each potential offer o of request i the utility value $UV_{i,o}$ is evaluated and returned within the offer to the originating domain that has received the user's request. The utility values are calculated by the user supplied utility function UF_i which can be extracted from the job and offer parameters. In addition the machine value $MV_{i,j}$ of the corresponding machine j can be included.

$$UV_{i,o} = UF_i(MV_{i,j}, o); \qquad MV_{i,j} = MF_j(i)$$

This machine value is derived from the machine objective function MF. The originating domain selects the offer with the highest utility value $UV_{i,o}$. In principle, this domain acts as an auctioneer. A more detailed explanation of the local offer generation is given in Figure 10.

Within the *Check Request* phase, it is determined if either the best offer will be automatically selected or if the user is going to select the best offer interactively among a given number of possible offers. In the same step, it is checked whether the user supplied budget is sufficient in order to process the job at the local machines in the domain. Additionally, it is determined whether the local resources meet the requirements of the request. Next, the necessary scheduling parameters are extracted. The parameters include the earliest start time of the job, the deadline, the maximum search time, the time until the resources will be reserved for the job (reservation time), the expected run time and the number of required resources. The utility function is another parameter which is applied in the further selection process.

If not enough resources can be found during the *Check Request* phase, but all other requirements can be satisfied by the local resources, a *multi-site scheduling* can be initiated to gather the additional resources which are necessary to satisfy the request. The ability of integrating such co-allocation strategies into this model will later be discussed in a more detail, see Section 4.4.

The next step *Search for idle intervals within the schedule* tries to find all idle time intervals within the requested time frame on the suitable resources. For a simple example assume a parallel computer with dedicated nodes as the resources. An example schedule is given in Figure 12. The black areas within the schedule are already allocated by other jobs. Now, a new incoming job requests three nodes and has a start time A, an deadline D and a run time less than $(C - B)$. First, idle time intervals are extracted for each node. Next, these elements are combined in order to find possible solutions. To this end, a list is created with triples of the form {time, node number, d}, where d equals (+1) if the node with the specified number is free at the examined time; d equals (-1) otherwise.
The generated list is used to find possible solutions as shown in the pseudo code in Figure 11.

```
list tempList;
time t = first time in sourceList;
LOOP: while(not enough offers found) {
    while(sourceList not empty) {
        tripleList = get and remove all next elements of sourceList with time t;

        test for all elements in tempList whether the difference between the beginning of
            the idle interval and time t is greater or equal to the run time of the job;

        if(number of elements in tempList, which fulfill the time
                condition, is greater or equal the needed number of nodes) {
            create an offer from the elements of the tempList;
       }

       according to value d add or substract all elements in tripleList from tempList;
       get next time t in sourceList;
    }
}
```

Fig. 11. Algorithm for Creating Possible Offers.

The given algorithm creates possible offers that are specified by start, end, run time and the requested number of nodes. Note, that we did not yet show how the offer is created from the node elements of idle time intervals in this list. This is achieved by the following algorithm. The algorithm has the goal to find a set of resources from a given list of idle time intervals. This has to take into account that idle times at resource elements may have different start and end times.

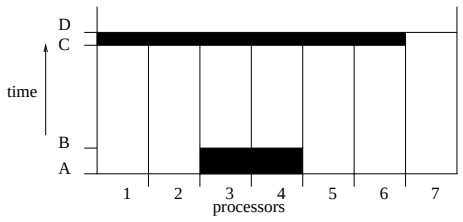

Fig. 12. Start Situation.

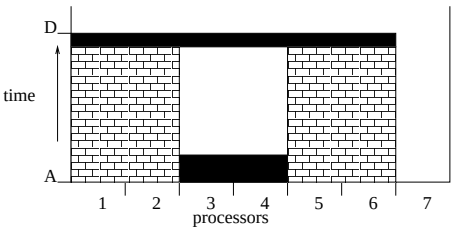

Fig. 13. Bucket 1.

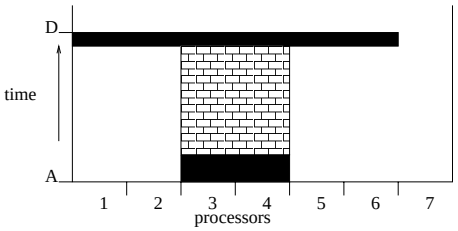

Fig. 14. Bucket 2.

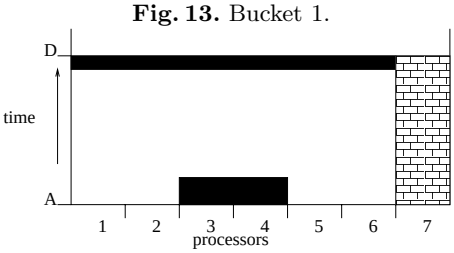

Fig. 15. Bucket 3.

The resulting possible node allocations are characterized by the earliest start and latest end time. To this end a derivation of a bucket sort is used. In the first step, all intervals with the same start time are collected in the same bucket. In the second step, for each bucket the elements with the same end time are collected in new buckets. At the end each bucket has a list of resources available between the same start and end time.

For the example above, the algorithm creates three bucket as shown in Figures 13, 14 and 15. After the creations of these buckets, suitable offers are generated either with elements from one bucket if the bucket includes enough resources or by combining elements of different buckets. When using elements from different buckets with potentially different start and end times, the maximum start and the minimum end time must be calculated. In our example only Bucket 1 can satisfy the requirements alone and therefore an offer can be built by using for instance Resources 1, 2 and 5.

In order to generate additional different offers, all buckets, which alone can satisfy a request with its own elements, are modified to contain one resource less than the required number. Afterwards, the previous process of offer generation is repeated again. If a bucket does not contain enough elements to satisfy a request, all elements of this bucket are taken into a new bucket. For our example this is true for the Buckets 2 and 3. If not enough offers have been found and no bucket is further available, it is checked whether the remaining number of elements in all buckets combined suffice the requested node number. If this is the case, the bucket elements are combined and the start and end times are adjusted to the maximum start and minimum end time. In our example, the set of solutions including combinations from Bucket 1 is: {{1,2,5}, {1,2,3}, {1,2,4}, {1,2,7}, {1,3,4}, {1,3,7}, {1,4,7}, {2,3,4}, {2,3,7}, {3,4,7}}.

After finishing the phase *Search for idle intervals within the schedule* from Figure 10, a course grain selection of one of these intervals takes place in the next step. In general, a large number of solutions could be created with only small changes for the start and end time and then selecting the interval with the highest utility value. Due to the time constraints of the scheduling algorithm, this is not feasible in practice. Therefore, we use a heuristic that first selects several initial combinations. The start and end times for these combinations are modified to improve the utility value. The combination with the highest utility value is selected as the resulting offer (in Phase "fine selection of an interval" in Figure 10). This approach can be parameterized by the number of steps denoting the number of different start and end times within the given time interval. Note that as we do not require the utility function to be monotone. Therefore, this selection process is a heuristic approach.

After finishing the algorithm in this last step, several possible offers have been generated.

We did not yet specify the utility functions of the machine provider and the user. In our implementation, any mathematical formula, using any valid time and resource variables, is supported. Overall, in our approach we select among the offers the solution with the highest value for the user's utility function. Note, that this process does not necessarily yield the global maximum among all possible offers. The linkage to the objective function of the machine provider is created by the price for the machine usage which equals the machine provider's utility function. The user can include this price in his utility function.

Fig. 16. Parameters for the Calculation of the Provider Utility Function.

The provider of the machine can formulate a utility function in which additional variables can occur that depend on the resulting schedule. Figure 16 shows variables that are used in our implementation. The variable *before* specifies the processor times in the schedule in which the corresponding resources (nodes) are unused before the job allocation. The variable *after* determines the processor times of unused resources after the job to the start of next job start on the according resources or to the end of the schedule. The variable *parallel* specifies the concurrently idle resource times during the allocation of the job. In a graphical depiction of a schedule as in Figure 16, this is the idle area parallel to the job for all nodes of the machine. The variable *utilization* specifies the utilization of the machine if the job is allocated. This is defined by the ratio of the sum of all allocated processor times and the whole available processor times from the current time instant to the end of the schedule.

4.4 Co-Allocation (Multi-Site Scheduling)

In the following we cover the situation when none of the local machines is alone able to compute the job. In that case, a multi-site allocation is sought by co-allocating resources from different machines. As shown in Figure 10, a domain might request resources from other domains. This process is only initiated if there is no single machine in any domain able to satisfy the request. The main problem of multi-site scheduling is to find several appropriate resources to execute the whole job in parallel. The missing resources are queried from other domains by additional requests

for fixed time frames. To this end, several different start times are tried within the possible time interval given in the job request. The process finishes if a solution is found or a time limit is reached.

We used this co-allocation mechanism in our evaluation for executing computational jobs spread over different machines. However, this approach can easily be extended for co-allocating and coordinating different resource types. The same method can be applied to make reservation of network bandwidth, storage or software in conjunction with a computational job. The need for more complex coordination can be found in workflows in which different job steps may have dependencies. Within the presented economic model, the capability to individually request certain resource offers can be used to build sophisticated Grid schedulers for such coordination tasks.

4.5 Simulation and Evaluation

We examine the performance of the economic scheduling method by assuming that the overall objective is actually the reduction of the overall response time. This allows the comparison to the conventional scheduling algorithms as presented in the previous section. For this economic scheduling model we have to use corresponding utility functions for minimizing the response time. However, we do not know yet which objective functions for machines and users achieve a low average weighted response time. Therefore, several utility functions have been examined in this work as a first starting point.

Resource Configurations We use the same configurations as presented in Section 3.1 in Table 1. Again, all configurations use a total of 512 resources.

For the simulations, 6 different provider objective functions were used. Here, we briefly describe the first provider **m**achine **f**unction MF_1 whose included terms are also used in the other functions:

$$NumberOfProcessors \cdot RunTime$$

calculates the processor times that the job is using within the schedule. The second term calculates the idle processor times before and after the job as well as the parallel idle times for all other resources within the local schedule (see Figure 16.):

$$before + after + parallel.$$

The last term of the formula is:

$$1 - parallel\_rel,$$

where $parallel\_rel$ describes the relation between the idle processor times in parallel to the job within the schedule (parallel) and the processor times actual used by the job. A small factor describes that the idle processor times in parallel are small in comparison to the job resource consumption. This leads to the following objective function MF_1 and its derivations MF_2 - MF_6:

$$\begin{aligned}
MF_1 &= (NumberOfProcessors \cdot RunTime \\
&\quad + after + before + parallel) \cdot (1 - parallel_rel) \\
MF_2 &= (NumberOfProcessors \cdot RunTime \\
&\quad + after + before + parallel), \\
MF_3 &= (NumberOfProcessors \cdot RunTime \\
&\quad + after + before) \cdot (1 - parallel_rel), \\
MF_4 &= (NumberOfProcessors \cdot RunTime \\
&\quad + parallel) \cdot (1 - parallel_rel), \\
MF_5 &= (NumberOfProcessors \cdot RunTime \\
&\quad + after + parallel) \cdot (1 - parallel_rel), \\
MF_6 &= (NumberOfProcessors \cdot RunTime \\
&\quad + before + parallel) \cdot (1 - parallel_rel).
\end{aligned}$$

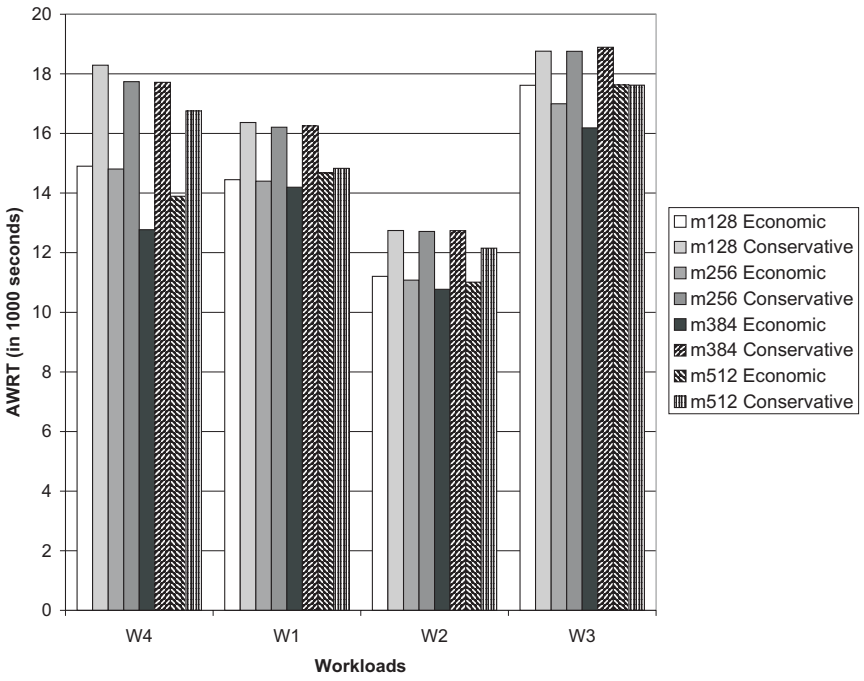

Fig. 17. Comparison Between Economic and Conventional Scheduling.

Job Configurations For the evaluation we use the same workloads as described in Table 2 of Section 3. Additionally, a utility function for each job is now necessary to represent the preferences of the corresponding user. To this end, the following 5 user **u**tility **f**unctions (UF) have been applied in our simulations. During a simulation, we assume that all users have the same utility function. Again, these are example utility functions to get first information on the impact on the scheduling performance. Further work is necessary for optimizing these functions for real applications.

The first user utility function prefers the earliest start time of the job. All processing costs are ignored.

$$UF_1 = (-StartTime).$$

The second user utility function only considers the calculation costs caused by the job.

$$UF_2 = (-JobCost).$$

The last user utility functions are combinations of the first two, but with different weights.

$$UF_3 = (-(StartTime + JobCost))$$
$$UF_4 = (-(StartTime + 2 \cdot JobCost))$$
$$UF_5 = (-(2 \cdot StartTime + JobCost)).$$

Results for the Economic Strategies Figure 17 shows a comparison of the average weighted response time for the economic method and for the conventional first-come-first-serve/backfilling scheduling system. For both systems the best achieved results have been selected. Note, that the used machine and utility functions differ between the economic simulations. The results show that for all used workloads and all resource configurations the economically based scheduling system

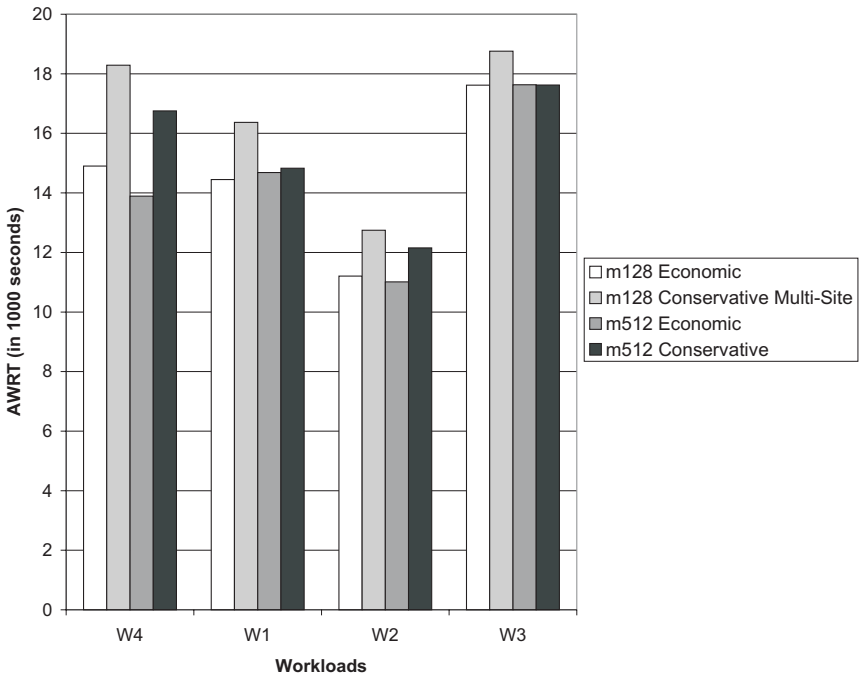

Fig. 18. Comparison Between Economic and Conventional Scheduling for the Resource Configurations *m128* and *m512* Using *MF1 - UF1*.

has the capability to outperform the conventional first-come-first-serve/backfilling strategy. Note that the economic scheduler is able to outperform backfilling as it is not restricted in job execution order. Figure 17 shows the best results for the economic scheduling system.

Figure 18 gives a comparison between the economic and the conventional scheduling system for only one machine/utility function combination. The used combination of MF_1 and UF_1 outperforms the conventional system for all used workloads and the configurations m128 and m512.

Certain combinations of machine and user utility functions can provide good results for different workloads. In the following, we compare different machine/utility functions which are exemplarily shown for the resource configuration m128. In Figure 19 the average weighted response time is given for all different machine functions in combination with utility function UF_3. The average weighted response time for the machine function MF_2 performs significantly better than all other machine functions. Here, the factor $1 - parallel\_rel$, which is used in all other machine functions, does not work well for this machine configuration. Instead it seems to be beneficial to use absolute values for the processor times, e.g. $(NumberOfProcessors \cdot RunTime + after + before + parallel)$. Unexpectedly, Figure 19 also shows that the intention to reduce the free areas within the schedule before a job starts (with attribute $before$) results in very poor average weighted response times (see the results for MF_1, MF_3, MF_6).

Utility function UF_1, which only takes the job start time into account, results in the best average weighted response time. In this case, no attention was paid to the resulting job cost. This means that no minimization of the idle processor times in parallel to the job is regarded. The utility functions which include this job cost yield inferior results in terms of the average weighted response times. The second best result originates from the usage of the utility function UF_3. In opposite to UF_1 the starting time and the job costs are equally weighted. A higher response time was achieved for all other utility combinations, in which either only the job costs (UF_2) or unbalanced weights for the starting time are used.

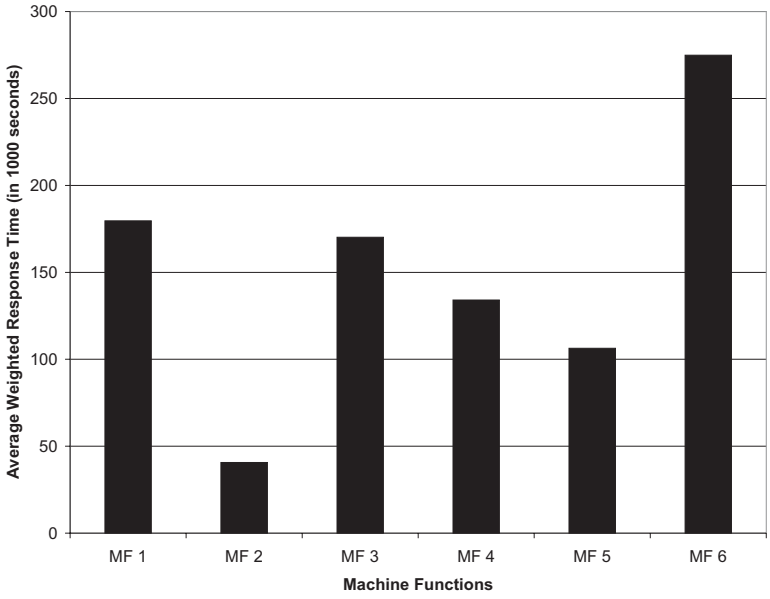

Fig. 19. The Resulting Average Weighted Response for Resource Configuration *m128*, Utility Function *UF3* and Several Machine Functions for Workload *W4*.

5 Conclusion

In this paper we distinguished between two different Grid scenarios. The first scenario resembles HPC Grids with a limited number of participating sites and a dedicated user community. The second scenario considers a Global Grid with potentially a large number of users and resource providers. We presented scheduling strategies for these scenarios and examined the performance by discrete event simulations for different workloads and machine configurations.

The results demonstrate that the used economical model can achieve results within the range of conventional algorithms in terms of the average weighted response time. However in comparison to the conventional methods, the economical method leaves a much higher flexibility in defining the desired resources. Also the problems of site autonomy, heterogenous resources and individual provider policies are solved by the structure of this economic approach. Moreover, the provider and user utility functions may be set individually for each job request. Additionally, features as co-allocation and multi-site scheduling over different resource domains are supported. Especially the advance reservation of resources is an advantage. In comparison to conventional scheduling systems there is instant feedback by the scheduler on the expected start time of a job already at submit time. Note, that the examined utility functions in the simulations are first approaches and leave room for further analysis and optimization. A more extensive parameter study for comprehensive knowledge on their influence on cost and execution time is necessary.

In general, it seems advantageous to consider market-oriented scheduling strategies for future Grid systems. The presented results show that these strategies can concur with conventional methods while providing a wide range of additional features that will be essential for Grids. A key element for establishing efficient scheduling strategies for Grids is the standardization of corresponding protocols and interfaces for a Grid scheduling architecture. Based on our research results, it seems likely that such an architecture can be designed without any restriction to a particular scheduling strategy. That is, it should be possible to build Grid markets in which different providers maintain individual local scheduling strategies. Similarly, it is unlikely that a single Grid scheduling strategy will fit all user requirements. But in such a Grid market there is

the flexibility to have individual Grid scheduling instances that cooperate with each other or with local resource management system by negotiating about agreement.

Current Grid scheduling research suffers from the limited knowledge about actual Grid users and workloads. Therefore, we limited our examinations in this work on existing workload traces from real computing sites and a corresponding scheduling objective. Future work will incorporate more extensive user and workload models as well as exploration on new Grid business models.

Further research is necessary to extend the presented model to incorporate network, data and storage as limited resources which have to be managed and scheduled as well. In this case additional service can be designed similar to a managed computing resource which provides information on offers or guarantees for possible allocations, e.g. quality-of-service features.

References

1. N. Bogan. Economic Allocation of Computation Time with Computation Markets. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 1994.
2. M. Brune, J. Gehring, A. Keller, and A. Reinefeld. Managing clusters of geographically distributed high-performance computers. *Concurrency - Practice and Experience*, 11(15):887–911, 1999.
3. R. Buyya, D. Abramson, and J. Giddy. An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications. In *The 2nd Workshop on Active Middleware Services (AMS 2000), In conjuction with Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 2000), Pittsburgh*. Kluwer Academic Press, Norwell, August 2000.
4. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing. *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14:1507–1542, November - December 2002.
5. K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Job Scheduling Strategies for Prallel Processing*, volume 1459 of *Lecutre Notes in Computer Science*, pages 62–68. Springer, 1998.
6. C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Enhanced algorithms for multi-site scheduling. In *Proceedings of the 3rd International Workshop on Grid Computing, Baltimore*, volume 2536 of *Lecture Notes in Computer Science*, pages 219–231. Springer, 2002.
7. C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. On advantages of grid computing for parallel job scheduling. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID2002)*, pages 39–46, Berlin, May 2002. IEEE Press.
8. C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. Effects of machine configurations on parallel job scheduling in computational grids. In *Proceedings of the 6th Workshop on Parallele Systeme und Algorithmen*, pages 169–179, Karlsruhe, April 2002. VDE.
9. C. Ernemann, V. Hamscher, and R. Yahyapour. Economic scheduling in grid computing. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *Lecture Notes in Computer Science*, pages 128–152. Springer, 2002.
10. C. Ernemann, V. Hamscher, and R. Yahyapour. Benefits of global grid computing for job scheduling. In *5th IEEE/ACM International Workshop on Grid Computing, in Conjunction with SuperComputing 2004, GRID 2004 Pittsburgh, PA, USA, November 8, 2004*, pages 374–379. IEEE Computer Society, November 2004.
11. C. Ernemann and R. Yahyapour. *Grid Resource Management - State of the Art and Future Trends*, chapter Applying Economic Scheduling Methods to Grid Environments, pages 491–506. Kluwer Academic Publishers, 2003.
12. 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.
13. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In *IPPS'97 Workshop: Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 1–34. Springer, April 1997.
14. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

15. V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *Proceedings of the 7th International Conference on High Performance Computing, HiPC-2000*, volume 1971 of *Lecture Notes in Computer Science*, pages 191–202, Bangalore, Indien, 2000. Springer.
16. 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*, volume 1162 of *Lecture Notes in Computer Science*, pages 27–40. Springer, 1996.
17. C. Kenyon and G. Cheliotis. *Grid Resource Management - State of the Art and Future Trends*, chapter Grid Resource Commercialization - Economic Engineering and Delivery Scenarios, pages 465–478. Kluwer Academic Publishers, 2003.
18. 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*, volume 949 of *Lecture Notes in Computer Science*, pages 295–303. Springer, 1995.
19. M. J. Litzkow and M. Livny. Condor — a hunter of idle workstations. In *Proceedings of the 8th IEEE International Conference Distributed Computing Systems*, pages 104–111, 1988.
20. M. Livny and R. 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.
21. U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, Nov 2003.
22. J.M. Schopf. *Grid Resource Management - State of the Art and Future Trends*, chapter Ten Actions When Grid Scheduling - The User as a Grid Scheduler, pages 15–23. Kluwer Academic Publishers, 2003.
23. U. Schwiegelshohn and R. Yahyapour. Analysis of first-come-first-serve parallel job scheduling. In *Proceedings of the 9th SIAM Symposium on Discrete Algorithms*, pages 629–638. SIAM, Philadelphia, January 1998.
24. U. Schwiegelshohn and R. Yahyapour. Improving first-come-first-serve job scheduling by gang scheduling. In *IPPS'98 Workshop: Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 180–198. Springer, March 1998.
25. U. Schwiegelshohn and R. Yahyapour. *Grid Resource Management - State of the Art and Future Trends*, chapter Attributes for Communication Between Grid Scheduling Instances, pages 41–52. Kluwer Academic Publishers, 2003.
26. J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY — LoadLeveler API Project. *Lecture Notes in Computer Science*, 1162:41–47, 1996.
27. B. Song, C. Ernemann, and R. Yahyapour. Modeling of parameters in supercomputer workloads. In U. Brinkschulte, J. Becker, D. Fey, K.-E. Großpietsch, C. Hochberger, E. Maehle, and T. Runkler, editors, *Proceedings of the Workshop on Parallel Systems and Algorithms (PASA) in conjunction with ARCS 2004: Organic and Pervasive Computing, Augsburg 26th of March 2004*, volume P-41 of *Lecture Notes in Informatics (LNI)*, pages 400–409. Gesellschaft für Informatik, Bonn, Köllen Druck+Verlag GmbH Bonn, March 2004.
28. B. Song, C. Ernemann, and R. Yahyapour. Parallel computer workload modeling with markov chains. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing: 10th International Workshop, JSSPP 2004 New York, NYC, USA, June 13, 2004*, Lecture Notes in Computer Science 3277, pages 47–62. Springer, October 2005.
29. Parallel workload archive. http://www.cs.huji.ac.il/labs/parallel/workload/, May 2006.
30. K. R. Grant T. W. Malone, R. E. Fikes and M. T. Howard. Enterprise: A market-like task scheduler for distributed computing environments. In *The Ecology of Computation*, volume 2 of *Studies in Computer Science and Artifical Intelligence*, pages 177–255, 1988.
31. P. Tucker and F. Berman. On market mechanisms as a software technique, December 1996.
32. C. A. Waldspurger, T. Hogg, B. Huberman, J. O. Kephart, , and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103– 117, 1992.
33. W. Walsh, M. Wellman, P. Wurman, and J. MacKieMason. Some economics of market-based distributed scheduling. In *In Eighteenth International Conference on Distributed Computing Systems*, pages 612–621, 1998.
34. F. Ygge. *Market-Oriented Programming and its Application to Power Load Management*. PhD thesis, Department of Computer Science, Lund University, 1998.