Economic Scheduling in Grid Computing

Carsten Ernemann, Volker Hamscher, and Ramin Yahyapour

Computer Engineering Institute, University of Dortmund, 44221 Dortmund, Germany,
(email: {carsten.ernemann,volker.hamscher,ramin.yahyapour}@udo.edu)

Abstract. Grid computing is a promising technology for future computing platforms. Here, the task of scheduling computing resources proves difficult as resources are geographically distributed and owned by individuals with different access and cost policies. This paper addresses the idea of applying economic models to the scheduling task. To this end a scheduling infrastructure and a market-economic method is presented. The efficiency of this approach in terms of response- and wait-time minimization as well as utilization is evaluated by simulations with real workload traces. The evaluations show that the presented economic scheduling algorithm provides similar or even better average weighted response-times as common algorithms like backfilling. This is especially promising as the presented economic models have additional advantages as e.g. support for different price models, optimization objectives, access policies or quality of service demands.

1 Introduction

Grid computing is expected to provide easier access to remote computational resources that are usually locally limited. Distributed computer systems are joined in such a grid environment (see [5, 12]), in which users can submit jobs that are automatically assigned to suitable resources. The idea is similar to metacomputing [20] where the focus is limited to compute resources. Grid computing takes a broader approach by including networks, data, visualization devices etc. as accessible resources [17, 11]. In addition to the benefit of access to locally unavailable resource types, there is also the expectation that a larger number of resources is available for a single job. This is assumed to result in a reduction of the average job response time. Moreover, the utilization of the grid computers and the job-throughput is likely to improve due to load-balancing effects between the participating systems.

Typically the parallel computing resources are not exclusively dedicated to the grid environment. Furthermore, they are usually not owned and maintained by the same administrative instance. Research institutes as well as laboratories and universities are examples for such resource owners. Due to the geographically distributed resources and the different owners the management of the grid environment becomes rather complex, especially the scheduling of the computational tasks. To this end, economic models for the scheduling are an adequate

way to solve this problem. They provide support for individual access and service policies to the resource owners and grid users. Especially the ability to include cost management into the scheduling will become an important aspect in future grid economy as anonymous users compete for resources.

In this paper, we present an architecture and an economical scheduling model for such grid environments. First examinations of the efficiency of this approach have been performed by simulations. The results are discussed in comparison to conventional scheduling algorithms that are not based on economic models. Note, that these classic methods are primarily optimized for response-time minimization.

The following sections are organized as follows. Section 2 gives a short overview on the background of grid scheduling and economic market methods. In Section 3 an infrastructure for a grid environment is presented that supports economic scheduling models as well as common algorithms as for instance backfilling. The economic scheduling method itself is described in Section 4. The simulation and the results for this scheduling method are shown in Section 5. The paper ends with a brief conclusion in Section 6.

2 Background

Scheduling is the task of allocating resources to problems over time. In grid computing these problems are typically computational tasks called jobs. They can be described by several parameters like the submission time, run time, the needed number of processors etc.

In this paper we focus only on the job scheduling part of a grid management infrastructure. A complete infrastructure has to address much more additional topics as e.g. network and data management, information collection or job execution. One example for a grid management infrastructure is Globus [10]. Note, that we examine scheduling for parallel jobs where the job parts can be executed synchronously on different machines. It is task of the grid scheduling system to find suitable resources for a job and to determine the allocation times. However, the actual transfer, execution, synchronization and communication of a job is not part of the grid scheduling system.

Until now mostly algorithms as e.g. FCFS and backfilling have been used for the scheduling task [8, 16]. These classic methods have been subject to research for a long time and have a well known behavior in terms of worst-case and competitive analysis. These algorithms have been used for the management of single parallel machines. In later implementations they were adapted for the application in grid environments [13, 6]. As already mentioned, the requirements on the scheduling method differs from single machine scheduling as the resources are geographically distributed and owned by different individuals. The scheduling objective is usually the minimization of the completion time of a computational job on a single parallel machine. Especially for grid applications other objectives have to be considered as cost, quality of service etc. To this end, other scheduling approaches are necessary that can deal better with different user objectives as

well as owner and resource policies. Here, naturally, economic models come into mind.

An overview on such models can be found in [2] and economic concepts have been additionally examined in the Mariposa project which is restricted to distributed database systems [22]. In comparison to other economic approaches on job scheduling (e.g. [27, 21]), our model supports varying utility functions for the different jobs and resources. Additionally, the model is not restricted to single parallel machines and allows further co-allocation of resources from different owners without disclosing policy information.

In this paper we just give a brief introduction on the background for our scheduling setting.

2.1 Market Methods

Market methods, sometimes called *Market oriented programming* in combination with Computer Science, are used to solve the following problems which occur in real scheduling environments ([4]):

- **The site autonomy problem** arises as the resources within the system are owned by different companies.
- **The heterogeneous substrate problem** that results from the fact that different companies use different resource management systems.
- **The policy extensibility problem** means that local management systems can be changed without any effects for the rest of the system.
- **The co-allocation problem** addresses the aspect that some applications need several resources of different companies at the same time. Market methods allow the combination of resources from different suppliers without further knowledge of the underlying schedules.
- **The online control problem** is caused by the fact that the system works in an online environment.

The supply and demand mechanisms provide the possibility to optimize different objectives of the market participants under the usage of costs, prices and utility functions. It is expected that such methods provide high robustness and flexibility in the case of failures and a high adaptability during changes.

Next, the definitions of *market*, *market method* and *agent* will be presented briefly.
A *market* can be defined as a virtual market or from an economical point of view as follows: *"Generally any context in which the sale and purchase of goods and services takes place."* [25]. The minimal conditions to define a virtual market are: *"A market is a medium or context in which autonomous agents exchange goods under the guidance of price in order to maximize their own utility."* [25]. The main aspect is that autonomous agents exchange voluntarily their goods in order to maximize their own utility.

A *market method* can be defined as follows: *"A market method is the overall algorithmic structure within which a market mechanism or principle is embedded."* [26]. It has to be emphasized that a market method is an equilibrium protocol and not a complete algorithm.

The definition of an *agent* can be found in [26]: *"An agent is an entity whose supply and demand functions are equilibrated with those of others by the mechanism, and whose utility is increased through exchange at equilibrium ratios."*.

It is now the question how the equilibrium can be obtained. One possible method is the application of *auctions*: *"An auction is a market institution with an explicit set of rules determining resource allocation and price on the basis of bids from the market participants."* [28]. More details about the general equilibrium and its existence can be found in [29].

2.2 Economic Scheduling in existing systems

Economic methods have been applied in various contexts. Besides the references explained in [2], we want to briefly mention some other typical algorithms of economic models.

WALRAS The *WALRAS* method is a classic approach by translating a complex, distributed problem into an equilibrium problem [1]. One of the assumptions is that agents do not try to manipulate the prices with speculation, which is called a *perfect competition*. 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 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.

The *WALRAS* method has been used for transportation problems as well as for processor rental. The transportation problem requires to transport different goods over an existing network from different start places to different end places. The processor rental problem consists of allocating one processor for different processes, while all processes have to pay for the utilization.

Enterprise Another application example for market methods is the *Enterprise* [24] system. Here, machines create offers for jobs to be run on these 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.

Under the premise of these methods, we present in the next sections our infrastructure and scheduling method for the grid job scheduling.

3 Infrastructure

The scheduling model presented in this paper has been implemented within the *NWIRE* (**N**et-**Wi**de-**Re**sources) management infrastructure which has been developed at our institute [19]. The general idea is that local management structures provide remote access to resources, which are represented by CORBA objects. The scheduling part is using those structures to trade resources between them. While staying locally controlled, the resources are offered throughout the connected management-systems.

To address the site autonomy problem, NWIRE structures the system into separate domains, that are constituted by a set of local resources and local management instances. Each so called MetaDomain is controlled by a MetaManager, as shown in Figure 1.

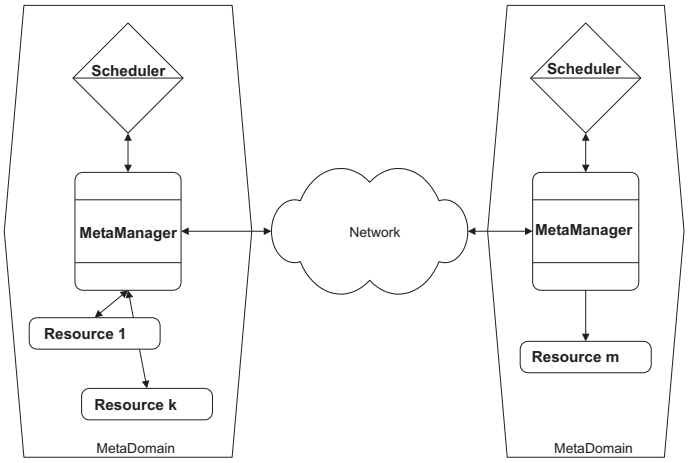

Fig. 1. Structure of NWIRE

This MetaManager administers the local resources and answers to local job requests. Additionally, this MetaManager consists of a local scheduler and acts as a broker/trader to other remote MetaDomains respectively their MetaManagers. That is the local MetaManager can offer local resources to other domains or tries to find suitable resource allocations for local requests.

The MetaManager can discover other domains by using directory services as well as exploring the neighborhood similar to peer-to-peer network strategies. If necessary, requests can be forwarded to the MetaManager of other domains. Parameters in the request are used to control depth and strategy of this search. Information on the location of specific resource types can be cached for later requests. Each MetaManager maintains a list with links to other dedicated MetaManagers. This list can be set up by the administrator to comply with logical or physical relationships to other domains, e.g. according to network or business

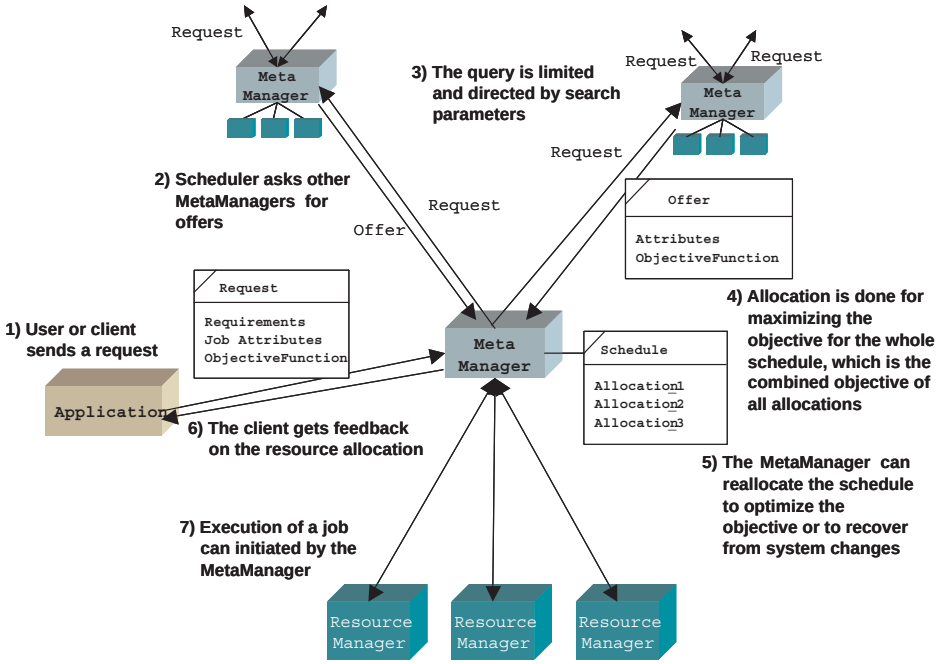

Fig. 2. Scheduling Steps

connections. Additionally, directory services can be introduced to find specific resource types. Information on remote resources can be cached and used to select suitable MetaManagers to which a request is forwarded.

This concept provides several advantages e.g. an increased reliability and fail-safety as the domains act independently. A failure at one site has only local impact as the overall network is still intact. Another feature is the ability to allow different implementations of the scheduling and the offer generation. According to the policy at an institution, the owner can setup an implementation that suits his needs best. Note, the policy on how offers for remote job requests are created does not have to be revealed.

This scheduling-infrastructure provides the base to implement different strategies for the scheduler. This also includes the ability to use conventional methods like for instance backfilling. Within the *NWIRE* system, this is achieved by using so called *requests* for the information exchange between the user and the components involved in the scheduling. The *request* is a flexible description of the conditions of a set of resources that are necessary for a *job*.

4 Economic Scheduling

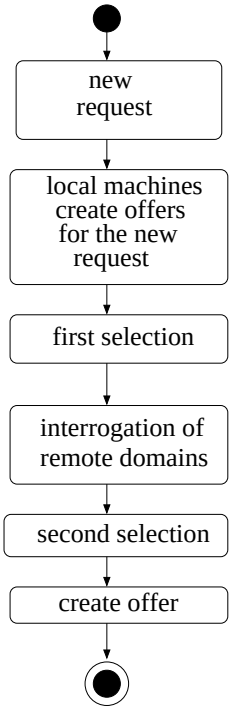

Fig. 3. General application flow.

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 3. In contrast to [3], our scheduling model does not rely on a single central scheduling instance. Moreover, each domain acts independently and may have different objective policies. Also the job requests of the users can have individual objective functions. The scheduling model has the task to combine these objectives to find the equilibrium of the market. This is a derivation of the previously presented methods of WALRAS and Enterprise.

In our scheduling model all users submit their job requests to the local MetaManager of the domain as shown in Figure 2. For example, the user specifies that his job requires 3 processors with certain properties as for instance the architecture. Additionally a utility function UF is supplied by the user. For instance the user in our example is interested in the minimization of the job start time, which can be achieved by maximizing the utility function $UF = (-StartTime)$.

The estimated job run-time is also given in addition to an earliest start and latest end time. Note, that a job is allocated for the requested run-time and is terminated if the job exceeds this time. If a job finishes earlier, the resulting idle time of resources can be allocated to later submitted jobs. These idle resources can be further exploited by introduction of a rescheduling step which has not been applied in this work. Rescheduling can be used to re-allocate jobs while maintaining the guarantees of the previous allocations. This can be compared with backfilling, although guaranteed allocations, e.g. due to remote dependencies by co-allocation, must be fulfilled. The rescheduling may require additionally requests for offers.

The request is analyzed by the scheduler of the receiving MetaManager. The scheduler creates, if possible, offers for all local machines. After this step, a first selection takes place where only the best offers are kept for further processing. According to the job parameters and the found offers, the request is forwarded to the schedulers of other domains. This is possible as long as the number of *hops* (search depth of involved domains) for this request is not exceeded and the time to live for this request is still valid. In addition none of the domains must have received this request before. The remote domains create new offers and send their best combinations back. If a job has been processed before no

further offers are generated. A second selection process takes place in order to find the best offers among the returned result of this particular domain.

Note, that this method is an auction with neither a central nor a decentral auctioneer. Moreover, the different objective functions of all participants are used for equilibration. For each potential offer o for request i the utility value $UV_{i,o}$ is evaluated and returned within the offer to the originating MetaDomain that received the user's request. The utility value is calculated by the user supplied utility function UF_i which can be formulated with the job and offer parameters. Additionally to this parameter set $\boldsymbol{P_u}$ the machine value $MV_{i,j}$ of the corresponding machine j can be included.

$$UV_{i,o} = UF_i(\boldsymbol{P_u}, MV_{i,j})$$
$$MV_{i,j} = MF_j(\boldsymbol{P_m})$$

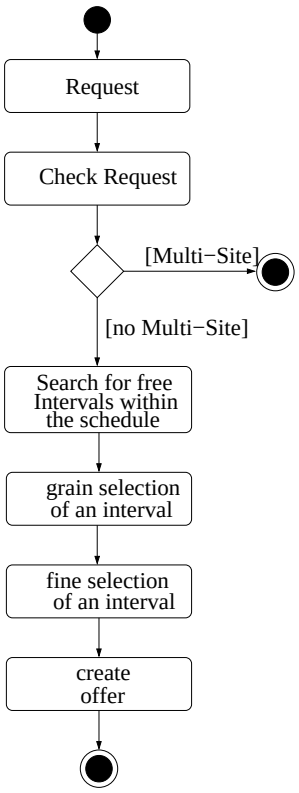

Fig. 4. Local offer creation.

The machine value results from the machine objective function MF which can depend on a parameter set $\boldsymbol{P_m}$.

The originating MetaManager selects the offer with the highest utility value $UV_{i,o}$. In principle this MetaManager serves the tasks of an auctioneer.

Next, we examine the local offer generation in more detail. To this end the application flow is shown in Figure 4.

Within the *Check Request* phase it is determined if either the best offer has to be automatically selected or if the user is going to select an offer interactively among a given number of possible offers.

In the same step the user's budget is checked whether it is sufficient in order to process the job at the local machines. The actual accounting and billing was not part of this study and requires additional work. Furthermore in this step, it is verified if local resources meet the requirements of the request. Next, the necessary scheduling parameters are extracted which are included in the request, e.g. the earliest start time of the job, the deadline (end time), 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. Another parameter is the utility function 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 fulfilled by the local resources, a *multi-site scheduling*

will be initiated. In this case additional and modified offers are requested from remote domains to meet in combination the original job requirements. This is an example of co-allocating resources from different owners.

The next step *Search for free intervals within the schedule* tries to find all free time intervals within the requested time frame on the suitable resources. As a simple example assume a parallel computer with dedicated processors as the resources. The example schedule is given in Figure 5. The black areas within the schedule are already allocated by other jobs. The job in our example requests three processors and has a start time A, an end time D and a run time less than $(C-B)$. First, free time intervals are extracted for each processor. Next, the free intervals of several processors are combined in order to find possible solutions. To this end, a list is created with triples of the form {time, processor number, +/-1} which means that the processor with the specified processor number is free (+1) or not free (-1) at the examined time.

The generated list is used to find possible solutions as shown in the following pseudo-code:

```
list tempList;
LOOP:while(generatedList not empty)
  {
    get the time t of the next element in the sourceList;

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

    if(number of elements in tempList, which fulfill the time
       condition, is bigger or equal the needed number of
       processors)
       {
         create offers from the elements of the tempList;
       }
     if(enough offers found)
       {
         finish LOOP;
       }
     add or substract the elements of the sourceList to or
     from tempList which have time entry t;
   }
```

The given algorithm creates potential offers that include e.g. start time, end time, run time and the requested number of processors as well as the user utility value ($UV_{i,o}$).

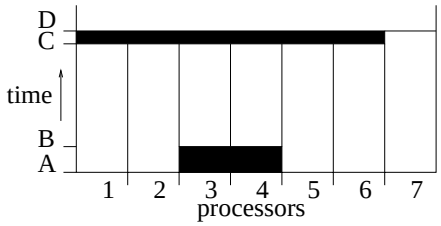

Fig. 5. Start Situation.

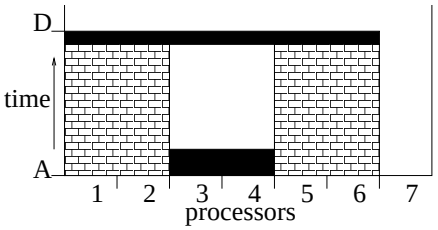

Fig. 6. Bucket 1.

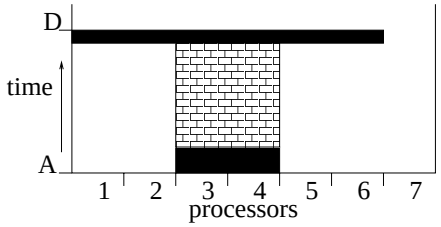

Fig. 7. Bucket 2.

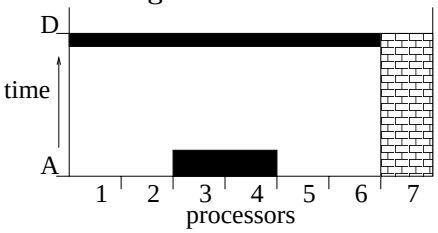

Fig. 8. Bucket 3.

Note, that it remains to be shown how the offer is created from the elements of this list. Such an algorithm will be presented in the following. The goal is to find areas of enough resources within the schedule for a given list of free time intervals. This has to take into account that resources possibly have different start and end times. The resulting areas 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 buckets as shown in Figures 6, 7 and 8. After the creation of buckets suitable offers are generated either with elements from one bucket if the bucket includes enough resources or by combining elements of different buckets. Additional care must be taken as elements from different buckets can have different start and end times. The maximum start and the minimum end time must be calculated. In our example only bucket 1 can fulfill the requirements alone and therefore an offer can be build e.g. with resources 1, 2 and 5.

In order to generate different offers a bucket for which an offer was only possible by its own elements is modified to contain one resource less than the required number. Afterwards, the process is continued. If yet not enough solutions are found and no further bucket can fulfill the request by itself as well as the number of remaining elements of all buckets is greater or equal to the requested resource number, new solutions are generated by combinations of bucket elements in regard to the intersecting time frames.

In our example, together with the solution build from bucket 1 the whole set of solutions would be: {{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 the end of the *Search for free intervals within the schedule* phase from Figure 4 a grain selection of one of these intervals takes place in the next phase. In principle a large number of solutions are possible by modifying the start and end time for the job in every combination and then selecting the interval with the highest utility value. In practice this is not applicable in regards to the run-time of the algorithm. Therefore a heuristic is used by selecting the combination having the highest utility value for the earliest start time. Next, the start and end time are modified to improve this utility value. The modification with the highest value is selected as the resulting offer during the phase "fine selection of an interval" in Figure 4.

A number of steps can be defined which specifies the number of different start and end times within the given time interval. Note, that the utility function is not constrained in terms of monotony. Therefore, the selection process above is heuristic.

After this phase the algorithm is finished and possible offers are generated.

The utility functions of the machine owner and the user have not been discussed yet. This method allows both of them to define their own utility function. In our implementation any mathematical formula, using any valid time and resource variables, is supported. Overall, the resulting value for the user's utility function is maximized. The linkage to the objective function of the machine owner is created by the price for the machine usage which equals the machine owner's utility function. The price may be included in the user's utility function.

The owner of the machine can build the utility function with additional variables that are first available after the schedule has been generated. Figure 9 shows variables that are used in our implementation. The variable *under* specifies the area in the schedule in which the corresponding resources (processors) are unused before the job allocation. *over* determines the area of unused resources after the job to the next job start on the according resources or to the end of the schedule. The variable *left_right* specifies the area on the left and right side of the job. The variable *utilization* specifies the utilization of the machine if the job is allocated. This is defined by the relation between the sum of all allocated areas to the whole available area from the current time instance to the end of the schedule.

Note, that the network has explicitly not been considered. Further work can easily extend the presented model to include network dependencies into the selection and evaluation process. For example, the network latency and bandwidth during job execution can be considered by parameterizing the job run-time during the scheduling.

However, the network is regarded in terms of resource partitioning and site autonomy. The presented model focuses on the cooperation scheme and economic scheduling scheme between the MetaManagers of independent domains. Herein,

a MetaManager can allocate jobs without direct control over remote resources and without the exposure of local control.

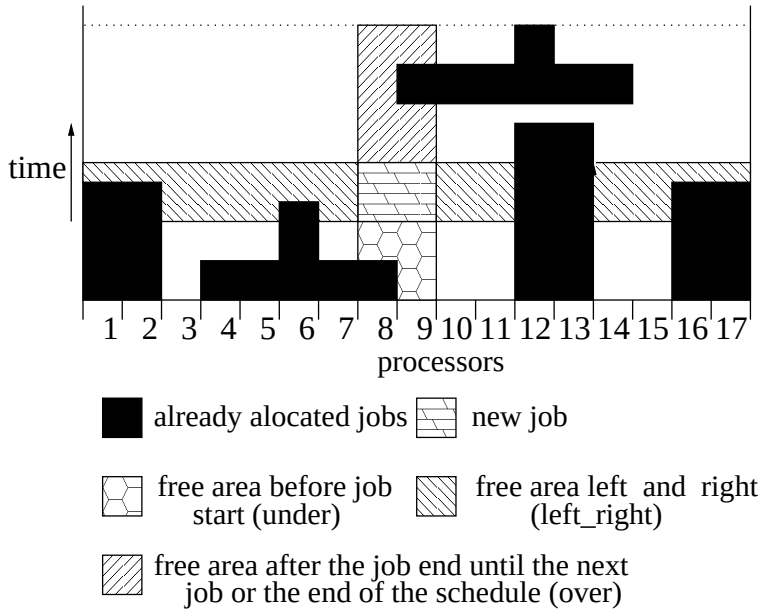

Fig. 9. Parameters for the calculation of the owner utility function.

5 Simulation and Evaluation

In this section the simulation environment is described. First, the resource configurations that are used for our evaluation are described followed by an introduction of the applied job model.

5.1 Resource Configurations

All three examined resource configurations have a sum of 512 processors in common. The configurations differ in the processor distribution on machines as shown in Table 1.

The configurations m128 and m256 are scenarios that resemble companies with several branch offices or a combination of universities. The configuration m384 characterizes a large data processing center which is connected to several smaller client sites. The configurations m128 and m256 are balanced in the sense of an equal number of processors at each machine. The configuration m384 in comparison is unbalanced. The resource configuration m512 serves as a reference with a single large machine executing all jobs.

identifier	configuration	maximum size	sum
m128	$4 \cdot 128$	128	512
m256	$2 \cdot 256$	256	512
m384	$1 \cdot 384 + 1 \cdot 64 + 4 \cdot 16$	384	512
m512	$1 \cdot 512$	512	512

Table 1. Used resource configurations.

In order to apply economic scheduling methods utility functions are required as mentioned before. Therefore, 6 different owner objective functions have been chosen for the first evaluation. Further extensive study is necessary to optimize the objective functions in regards to better results. The first one describes the most general owner utility function from which all others are derived. The owner machine function MF_1 consists of several terms. The first term:

$$NumberOfProcessors \cdot RunTime$$

calculates the area that the job is using within the schedule. The second term calculates the free areas before and after the job as well as the parallel idle time for the other resources within the local schedule (see Figure 9):

$$over + under + left\_right.$$

The last term of the formula is:

$$1 - left\_right\_rel,$$

where $left\_right\_rel$ describes the relation between the free areas to the left and right of the job within the schedule (left_right) and the area actual used by the job. A small factor describes that the free areas on both sides are small in comparison to the job area. This leads to the following objective function MF_1 and its derivations MF_2 - MF_6:

$$\begin{aligned}
MF_1 &= (NumberOfProcessors \cdot RunTime + over + under + left_right) \\
&\quad \cdot (1 - left_right_rel), \\
MF_2 &= (NumberOfProcessors \cdot RunTime + over + under + left_right), \\
MF_3 &= (NumberOfProcessors \cdot RunTime + over + under) \cdot (1 - left_right_rel), \\
MF_4 &= (NumberOfProcessors \cdot RunTime + left_right) \cdot (1 - left_right_rel), \\
MF_5 &= (NumberOfProcessors \cdot RunTime + over + left_right) \cdot (1 - left_right_rel), \\
MF_6 &= (NumberOfProcessors \cdot RunTime + under + left_right) \\
&\quad \cdot (1 - left_right_rel).
\end{aligned}$$

5.2 Job Configurations

Unfortunately, no real workload is currently 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 an IBM RS6000/SP parallel computer with 430 nodes. For more details on the traces and the configuration see the description of Hotovy [14]. The workload is available from the standard workload archive [23].

In order to use these traces for this study it was necessary to modify the traces 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 modeling 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, our configurations consist of 512 nodes. Nevertheless, the traces consist of enough workload to keep a sufficient backlog on conventional scheduling systems (see [13]). The backlog is the amount of 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. Note, that backlog analysis is only possible for the conventional scheduling algorithms. The economic method does not use a queue as the job allocation is directly scheduled after submission time.

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: 2.

identifier	description
10_20k_org	An extract of the original CTC traces from job 10000 to 20000.
30_40k_org	An extract of the original CTC traces from job 30000 to 40000.
60_70k_org	An extract of the original CTC traces from job 60000 to 70000.
syn_org	The synthetically generated workload derived from the CTC workload traces.

Table 2. The used workloads

The synthetic workload is very similar to the CTC data set, see [15]. It has been generated to prevent that singular effects in real traces, e.g. machine down times, do not affect the accuracy of the result. Also the usage of 3 extracts of the real traces are used to get information on the consistency of the results for

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

The same workloads have been applied for the simulations with conventional scheduling systems in [13, 6]. This allows the comparison of economic systems in this work to the non-economic scheduling systems in [13, 6, 7].

Additionally, a utility function for each job is necessary in economic scheduling to represent the preferences of the corresponding user. To this end, the following 5 user utility functions (UF) have been applied for our first evaluations.

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.

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

5.3 Results

Discrete event-based simulations have been performed according to the previously described architecture and settings.

Figure 10 shows a comparison of the average weighted response time for the economically based and for the conventional first-come-first-serve/backfilling scheduling system. The average weighted response time is the sum of the corresponding run and wait times weighted by the resource consumption which is the number of resources multiplied with the job execution time. Note that the mentioned weight prevents any prioritization of small over wider jobs in regard to the average weighted response time if no resources are left idle [18]. 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.

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 for all used workloads and all resource configurations that the economically based scheduling system has the capability to outperform the conventional first-come-first-serve/backfilling strategy.

Backfilling can be outperformed as the economic scheduling system is not restricted in the job execution order. Within this system a job, that was submitted after another already scheduled job, can be started earlier, if corresponding

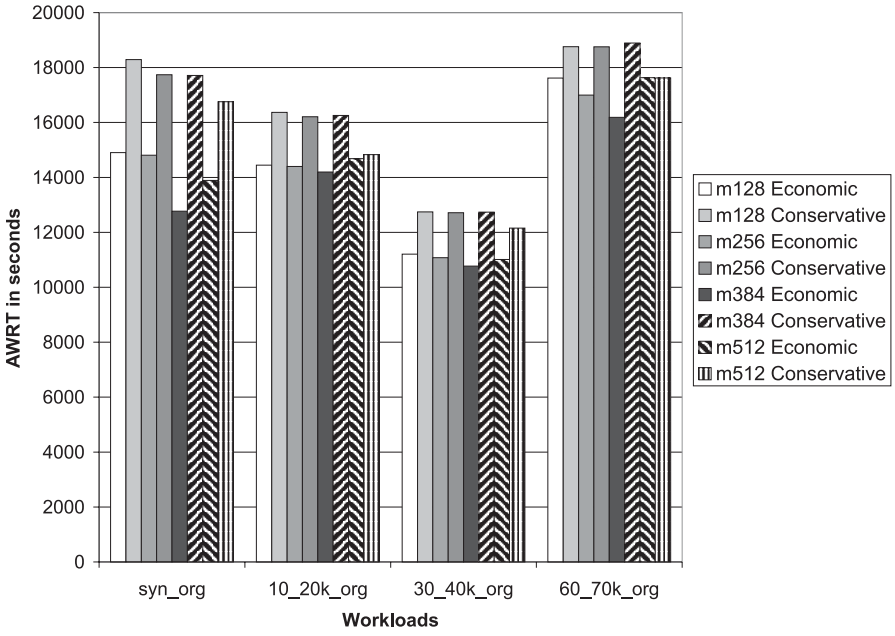

Fig. 10. Comparison between Economic and Conventional Scheduling.

resources can be found. The conventional backfilling strategy used with the first-come-first-serve algorithm ([16]) can only start jobs earlier if all jobs that were transmitted before are not additionally delayed. The EASY backfilling lowers this restriction to not delay the first job in the queue ([9]) does not result in a better performance. The restriction of out-of-order execution in backfilling prevents job starvation. The economic method does not encounter the starvation problem as the job execution is immediately allocated after submission.

Figure 10 only shows the best results for the economic scheduling system. Now, in Figure 11, a comparison between the economic and the conventional scheduling system for only one machine/utility function combination is presented.

The used combination of MF_1 and UF_1 leads to scheduling results that can outperform the conventional system for all used workloads and configurations m128 and m512. Note, that the benefit of the economic method was achieved by applying a single machine/utility function combination for all workloads. This indicates that suitable machine/user utility functions can provide good results for various workloads.

Figure 12 presents the AWRT combined with the average weighted wait time (AWWT) using the same weight selection. In all cases the same resource configuration as well as the same machine/utility function combination are used. The time differences between the simulations for both resource configurations

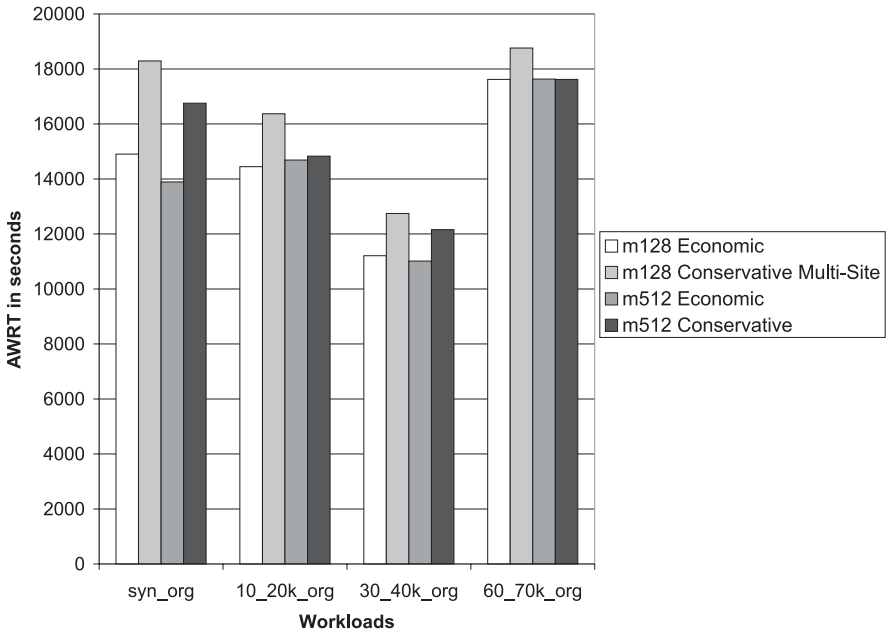

Fig. 11. Comparison between Economic and Conventional Scheduling for the Resource Configurations m128 and m512 using MF_1 - UF_1.

are small. This shows that the algorithm for multi-site scheduling (for resource configuration m128), although it is more complex, does not result in a much worse response time in comparison to a single machine. Note, that multi-site execution is not penalized by an overhead in our evaluation. Therefore, the optimal benefit of job splitting is examined and only the capability of supporting multi-site in an economic environment over remote sites is regarded. Here, effects of splitting jobs may even improve the scheduling results.

Figure 13 demonstrates that the average weighted response as well as the average weighted wait time do not differ significantly between the different resource configurations. In this case, the machine configurations prove limited impact on the effect on multi-site scheduling. Here, the overall number of processors is of higher significance in our economic algorithm. Configurations with bigger machines have smaller average weighted response times than configurations with a collection of smaller machines.

The influence of using different machine/utility function combinations for a resource set is shown in Figure 14. Here, the squashed area (the sum of the products of the run time and the number of processors) is given for different resource configuration. The variant m128 is balanced in the sense of having equal sized machines. The desired optimal behavior is usually an equal balanced workload distribution on all machines.

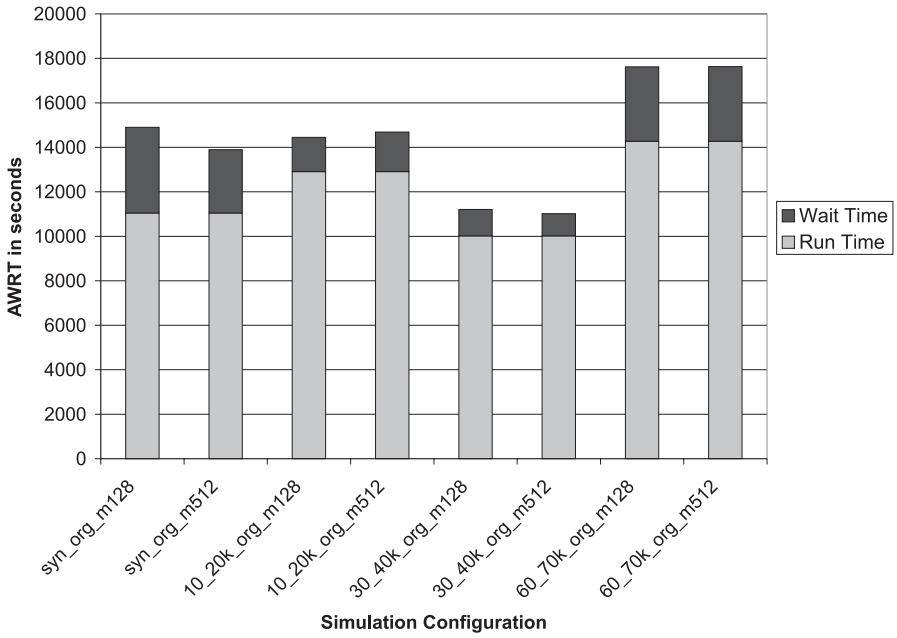

Fig. 12. AWRT and AWWT for m128 and m512 using several workloads, machine function MF_1 and utility function UF_1.

The combination of (MF_1, UF_1) leads to a workload distribution where the decrease of the local squashed area is nearly constant between the machines ordered by their number as shown in Figure 14. The maximum difference between the squashed areas is about 18%.

In the second case, the combination (MF_1, UF_2) presents a better outcome in sense of a nearly equally distributed workload.

The third function combination (MF_2, UF_2) leads to an unbalanced result. Two of the machines execute about 67% of the overall workload and the two remaining machines the rest.

Simulation results are shown for keeping the same machine/utility function combinations in Figure 15. The combination of (MF_1, UF_2) does not perform very well in terms of the utilization as all machines achieve less than 29%. This indicates in combination with Figure 14 that a well distributed workload corresponds with a lower utilization. The combination of (MF_1, UF_1) leads to a utilization between 61% and 77% on all machines. The third examined combination (MF_2, UF_2) shows a very good utilization of two machines (over 85%) and a very low utilization on the others (under 45%). In this case the distributed workloads correlates with the utilization of the machines.

After the presentation of the distributed workload and the corresponding utilization the AWWT and AWRT, shown in Figure 16 clearly indicates that only

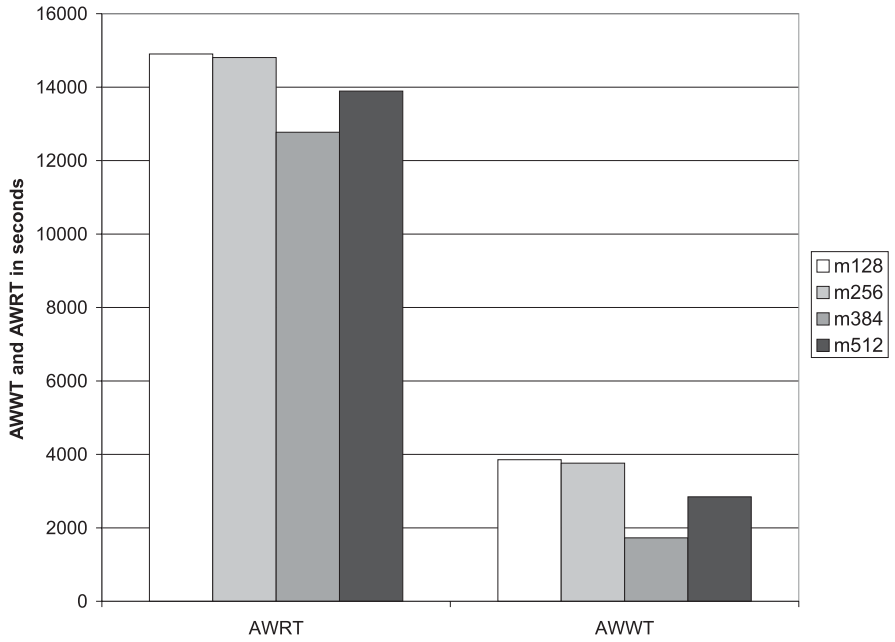

Fig. 13. AWRT and AWWT for all Resource Configurations and the syn_org workload in combination with MF_1 - UF_1.

the function combination (MF_1, UF_1) leads to reasonable scheduling results. The results from Figures 14,15 and 16 demonstrate that different machine/utility function combinations may result in completely different scheduling behaviors. Therefore an appropriate selection of these functions is important for an economic scheduling system.

In the following the comparison of different machine/utility functions is shown for the resource configuration m128. In Figure 17 the average weighted response time is drawn for all different machine function 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 $1 - left\_right\_rel$, which is used in all other machine functions, does not work well for this machine configuration. It seems to be beneficial to use absolute values for the areas instead, e.g. ($NumberOfProcessors \cdot RunTime + over + under + left\_right$). Unexpectedly, Figure 17 also shows that the intended reduction the free areas within the schedule before the job starts, with attribute *under*, results in very poor average weighted response times (see the results for MF_1, MF_3, MF_6).

As machine function (MF_2) provided significantly better results, different user utility functions are compared in combination with MF_2 in Figure 18.

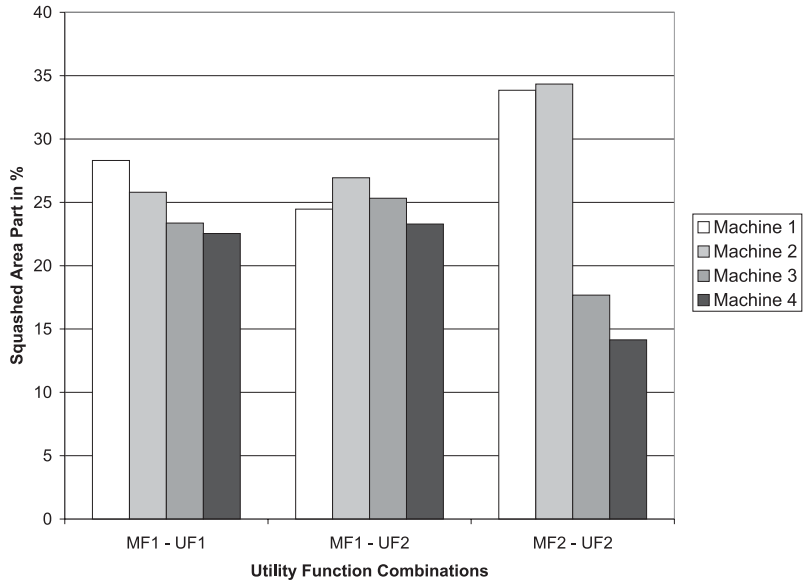

Fig. 14. The used Squashed Area of simulations with m128 and syn_org using different machine and utility functions.

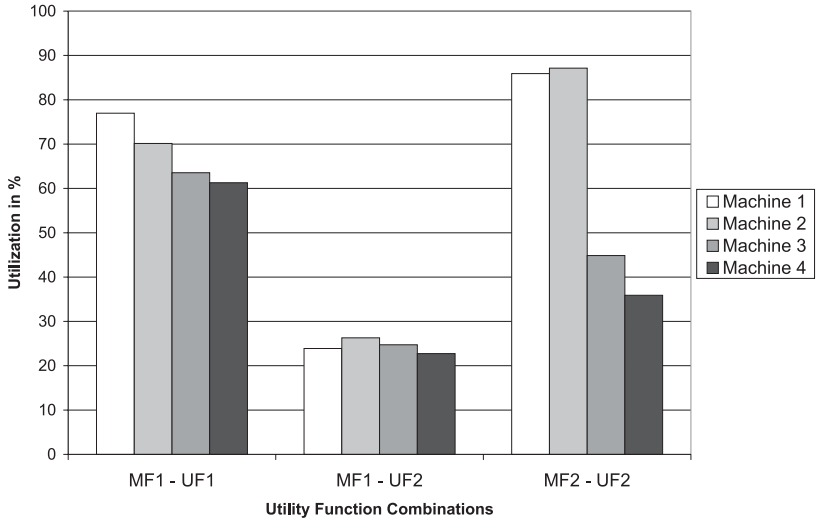

Fig. 15. The resulting utilization of simulations with m128 and syn_org using different machine and utility functions.

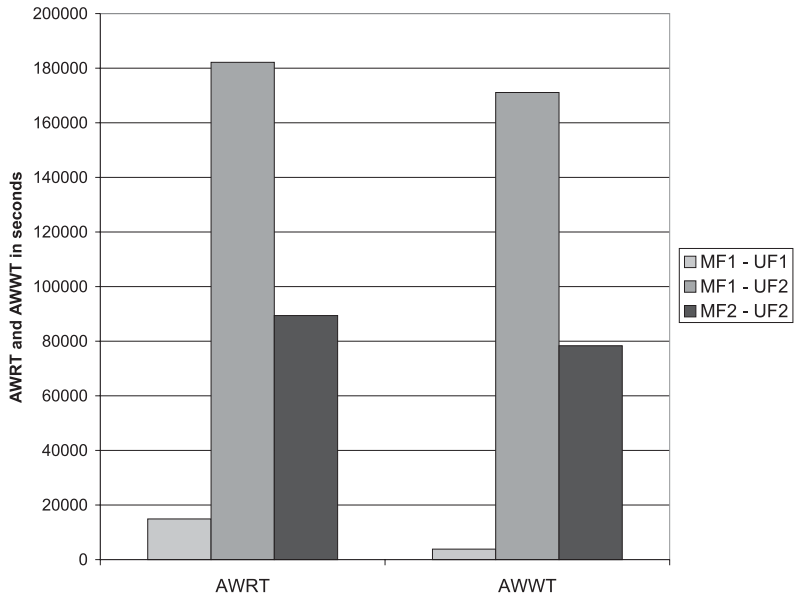

Fig. 16. The resulting average weighted response and wait times of simulations with m128 and syn_org using different machine and utility functions.

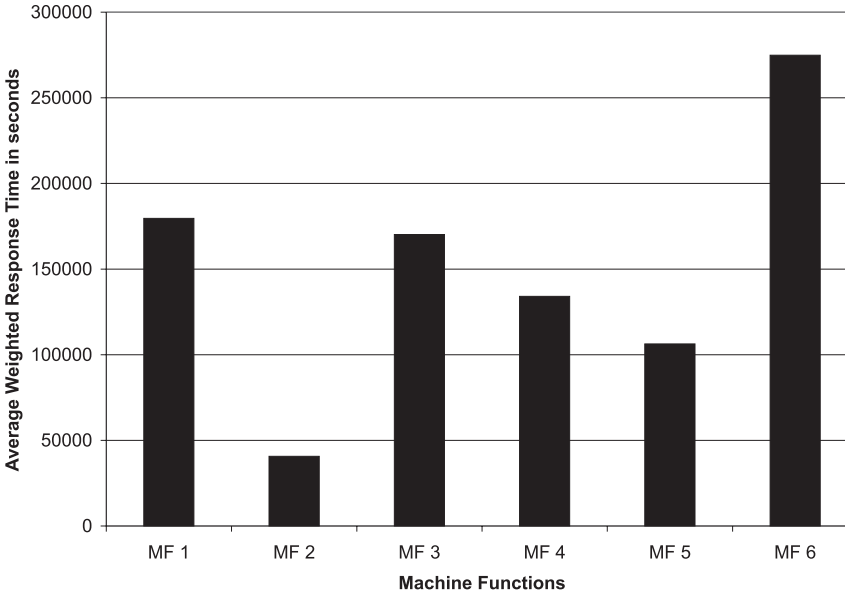

Fig. 17. The resulting average weighted response for resource configuration m128, utility function UF_3 and several machine functions.

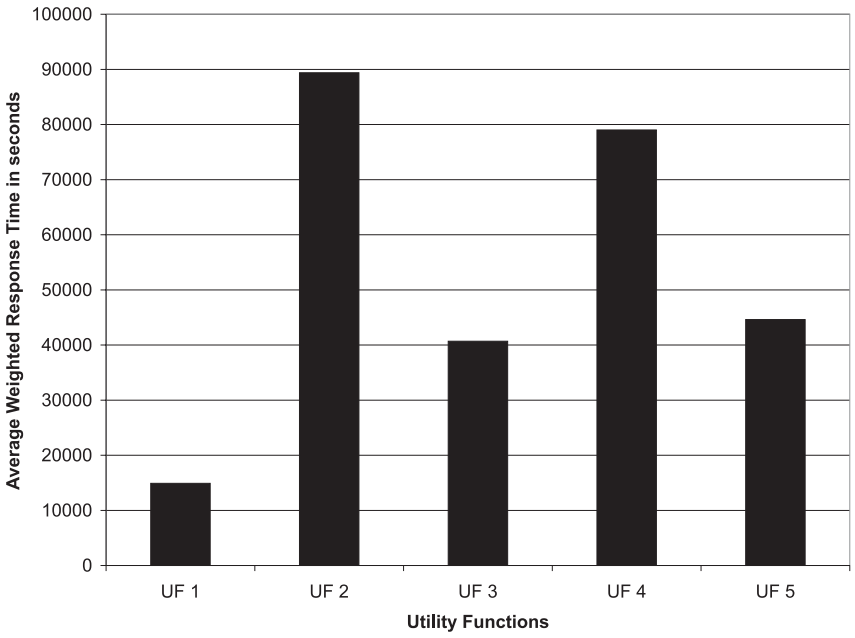

Fig. 18. The resulting average weighted response for resource configuration m128, machine function MF_2 and several utility functions.

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. For our selection of the machine objective function this means that minimization of the free areas around the job is not regarded. The utility functions that include this job cost deliver 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. All other utility combinations in which either only the job costs (UF_2) or unbalanced weights for the starting time and the job costs are used, lead to higher response times.

Note that the execution time of the simulations on a SUN-Ultra III machine varied according to the chosen machine and user utility functions. For an example the scheduling of 10000 jobs required about 1 hour, which means that the scheduling of one job lasts about one second on average. Nevertheless, this highly depends on the number of available resources. In an actual implementation the search time can be limited by a parameter given by the user or chosen by a heuristic based on job length and/or job arrival rate.

6 Conclusion

In this paper we presented an infrastructure and an economic scheduling system for grid environments. The quality of the algorithm has been examined by discrete event simulations with different workloads (4, each with 10.000 jobs), different machine configurations (4, each with a sum of 512 processors) and several parameter settings for owner and user utility functions.

The results demonstrate that the used economical model provides results in the range of conventional algorithms in terms of the average weighted response time. In comparison, the economical method leaves a much higher flexibility in defining the desired resources. Also the problems of site autonomy, heterogenous resources and individual owner policies are solved by the structure of this economic approach. Moreover, the owner and user utility function 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 possible advance reservation of resources is an advantage. In comparison to conventional scheduling systems there is instant feedback by the scheduler on the expected execution time of a job already at submit time. Note that conventional schedulers based on list scheduling as e.g. backfilling can provide estimates or bounds on the completion time. However, the economic method presented in this paper leads to a specific allocation in start and end-time as well as the resource. Guarantees can be given and maintained if requested. This includes the submission of jobs that request a specific start and end-time which is also necessary for co-allocating resources.

Note, that the examined utility functions in the simulations are first approaches and leave room for further analysis and optimization. Nevertheless, the results presented in this paper indicate that an appropriate utility function for a given resource configuration delivers steady performance on different workloads.

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

A more extensive parameter study for comprehensive knowledge on their influence on cost and execution time is necessary. To this end, future work can analyze scenarios in which different objective functions are assigned to each domain. Also the effect of a larger number of machines and domains in the grid must be evaluated.

The presented architecture in general provides support for re-scheduling, that means improving the schedule by permanently exploring alternative offers for existing allocations. This feature should be examined in more detail for optimizing the schedule as well as for re-organizing the schedule in case of a system or job failure.

References

1. N. Bogan. Economic allocation of computation time with computation markets. In *Master Thesis*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 1994.
2. 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)*, May 2002(accepted for publication).
3. R. Buyya, J. Giddy, and D. Abramson. An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications. In *The Second Workshop on Active Middleware Services (AMS 2000), In conjuction with Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 2000)*, Pittsburgh, USA, August 2000. Kluwer Academic Press.
4. 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 Verlag, 1998.
5. European grid forum, http://www.egrid.org, August 2002.
6. 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 International Symposium on Cluster Computing and the Grid (CC-GRID 2002), Berlin*, pages 39–46, 2002.
7. C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. On Effects of Machine Configurations on Parallel Job Scheduling in Computational Grids. In *International Conference on Architecture of Computing Systems, (ARCS 2002)*, pages 169–179. VDE-Verlag, April 2002.
8. D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In D.G. Feitelson and L. Rudolph, editors, *IPPS'97 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 1–34. Springer–Verlag, Lecture Notes in Computer Science LNCS 1291, 1997.
9. D.G. Feitelson and A.M. Weil. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In *Procedings of IPPS/SPDP 1998*, pages 542–546. IEEE Computer Society, 1998.
10. I. Foster and C. Kesselman. Globus: A metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.
11. I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
12. The Grid Forum, http://www.gridforum.org, August 2002.
13. 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.
14. 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.
15. J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the Design and Evaluationof Job Scheduling Systems. In D.G. Feitelson and L. Rudolph, edi-

tors, *IPPS/SPDP'99 Workshop: Job Scheduling Strategies for Parallel Processing*. Springer–Verlag, Lecture Notes in Computer Science, 1999.
16. 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.
17. 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.
18. 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.
19. U. Schwiegelshohn and R. Yahyapour. Resource Allocation and Scheduling in Metasystems. In P. Sloot, M. Bibak, A. Hoekstra, and B. Hertzberger, editors, *Proceedings of the Distributed Computing and Metacomputing Workshop at HPCN Europe*, pages 851–860. Springer–Verlag, Lecture Notes in Computer Science LNCS 1593, April 1999.
20. L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.
21. I. Stoica, H. Abdel-Wahab, and A. Pothen. A Microeconomic Scheduler for Parallel Computers. In D.G. Feitelson and L. Rudolph, editors, *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 200–218. Springer–Verlag, Lecture Notes in Computer Science LNCS 949, 1995.
22. M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A Wide-Area Distributed Database System. *VLDB Journal*, 5(1):48–63, 1996.
23. Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, August 2002.
24. 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.
25. P. Tucker. Market Mechanisms in a Programmed System. Department of Computer Science and Engineering, University of California, 1998.
26. P. Tucker and F. Berman. On market mechanisms as a software technique, 1996.
27. 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.
28. 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.
29. F. Ygge. *Market-Oriented Programming and its Application to Power Load Management*. PhD thesis, Department of Computer Science, Lund University, 1998.