Market-based Resource Allocation for Grid Computing: A Model and Simulation

Jacek Gomoluch and Michael Schroeder
Dept. of Computing, City University, London, UK
{ck655,msch}@soi.city.ac.uk

Abstract

Resource allocation is an important aspect of Grid computing. One approach uses market mechanisms to allocate resources. In this paper, we review the literature on market-based resource allocation for Grid computing classifying approaches as model- or state-based and pre-emptive or non-pre-emptive. Many of the existing market-based approaches take it for granted that markets are an improvement. We investigate under which circumstances market-based resource allocation by continuous double auctions and by the proportional share protocol, respectively, outperforms a conventional round-robin approach. To answer this question, we develop and justify a model for clients, servers and the market, and present simulation results. The factors which are studied include the amount of load in the system, the number of resources, different degrees of resource heterogeneity, and communication delays.

1 Introduction and Background

Recently, there has been much interest in *Computational Grids*, which provide transparent access to large-scale distributed computational resources. One important problem in such environments is the efficient allocation of computational resources. Over the past years, economic approaches to resource allocation have been developed [1] and the question arises whether they can be applied to resource allocation on the Grid. They satisfy some basic requirements for a Grid setting as they are naturally decentralised, as decisions about whether to consume or provide resources are taken locally by the clients or service providers, as the use of currency provides incentives for service providers to contribute resources, and as clients have to act responsibly and cannot afford to waste resources due to their limited budget. Indeed, a number of systems [2, 3, 4, 5, 6, 7, 8] have been built using a market mechanism to allocate the computational resources. However, all of them make the inherent assumption that a market-based approach is per se better, which is adhoc, as the allocation depends on many factors besides demand and supply, such as communication delays, bandwidth, server speeds, etc. Therefore, this paper investigates under which circumstances market mechanisms can lead to improvements. We define a model of the involved actors, marketplace, and protocols, ensuring that assumptions made are backed by observed and well-established distributions. Then we use discrete-event simulations to implement our model and to compare the performance of different resource allocation protocols. This enables us to model almost any type and number of resources. Parameters determining message delays, processing delays, task creation, number and speed of servers, etc. can be adjusted providing for a rich parameter space.

Before we introduce our model and results, let us review existing approaches. We can distinguish resource allocation strategies according to two criteria:

State-based vs Model-based: Are the allocations based on a current snapshot of the system state (*state-based*), which is expensive to obtain, or on a model, which predicts the system state and which may be inaccurate (*model-based* or *predictive*)?

Pre-emptive vs Non-pre-emptive: Are tasks assigned to hosts once (*non-pre-emptive*) and then stay there, or can they migrate if it turns out at a later stage that it is advantageous to leave the machine (*pre-emptive*)?

State-based, Non-Pre-emptive Strategies. State-based, non-pre-emptive strategies are easy to implement and can lead to good results. Therefore they are widely adopted. Market mechanisms provide a way of representing the system state and balancing load: they value resources and achieve an efficient match of supply and demand. While some systems use only a price and match offers and bids, others employ more sophisticated auction protocols [9]. Spawn [2], POPCORN [3], and CPM [5] are examples of systems which employ auctions. Spawn and CPM are de-centralised, and POPCORN and CPM also deal with resource accounting. Dynasty [4] pursues a different approach: avoiding the communication overhead of auctions, it uses brokering without any

ongoing negotiation. Prices are periodically fixed and there are fees for migration and data transport services. To achieve efficiency, the system is organised as a hierarchy of brokers, which exchange information in both directions. None of the above systems allows *true* task migration. However, Spawn and Dynasty, which deal with the allocation of divide&conquer applications, allow tasks to send subtasks to remote machines.

State-based, Pre-emptive Strategies. If the environment is very dynamic, it may be advantageous for a task to migrate elsewhere after being launched. Operating systems researchers already investigated how the allocation of resources can be optimised with mobility [10]. Mobile agents add a further degree of flexibility: Tasks become agents and can decide themselves, when and where to move to, with a global pattern of load-balancing emerging. In [7] a system is presented, which provides market-based resource control for mobile agents. To allocate resources to the agents the system uses electronic cash, a banking system and a set of resource managers. The pricing mechanism is a sealed-bid second price auction.

Model-based Strategies. Model-based approaches to resource allocation are much rarer, as they involve two very challenging problems: how to obtain an initial model and how to adapt the model as time passes. In the area of operating systems, some researchers explored this approach and used distributions of CPU load and expected process lifetime to decide if and when to migrate tasks [10]. A model-based approach is also used in Challenger [8]. It implements load-balancing with a market approach - however, without money. When a job is created, a 'request for bids' containing its priority value and information which can be used to estimate its duration is sent to the agents in the network. These make bids giving the estimated time to complete that job on their machine. Important parameters, which have a major impact on the system performance, are the message delay and errors in estimating the job's completion time. Learning behaviour has been introduced in order to deal with these problems. Another model-based strategy is used by the Nimrod-G Resource Broker [6]. Nimrod-G is a resource management system for scheduling computations on globally distributed resources with varying quality of service. The system is an economic-driven environment which supports various market-based protocols such as the commodity market model, posted pricing, and bargaining. It predicts the future performance of the resources in the system by resource capability measurements and load profiling.

All of the described systems implement a market-based strategy, often with the objective to improve performance, sometimes merely to try the market approach.

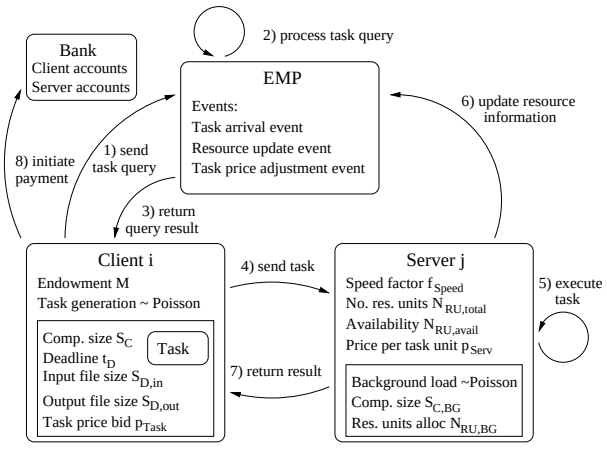

Figure 1: Model of the marketplace

For any such approach it is important to underpin any implementation by theoretical considerations and simulations justifying a market-based approach. Market-mechanisms involve more computation and communication than e.g. a simple round-robin allocation. On the other hand, round-robin may not handle heterogeneity of resources (speed, bandwidth, etc.) well. As a consequence, it is important to examine under which circumstances market-based approaches perform better. To answer this question, we develop a simulation model including actors and protocols and explore the simulation determining broad constraints for when markets are an improvement. For simplicity, we only consider state-based, non-pre-emptive strategies.

2 Actors and Model

Our model represents an electronic marketplace for distributed computational resources. As shown in Figure 1 there are three main actors in the model, which are assumed to be distributed over the Internet: the Clients, the Servers and the Electronic Marketplace (EMP). Clients generate tasks which require computational resources for their execution. The Servers provide these resources: they advertise and sell them at the Electronic Marketplace.

Clients. There is a fixed number of Clients N_{Client} in the system. Each Client generates tasks at a rate which is modelled by a Poisson arrival process. Poisson processes have been chosen, because they are suitable for describing user session arrivals on the Internet [11]. A Poisson arrival process has an exponential inter-arrival distribution. Its density function is given by $f(\tau) = \lambda \cdot e^{-\lambda\tau}$, where λ is the inverse of the mean time

between two task creations.

Tasks. A task which is created by a Client is characterised by the size of its computation S_C (in task units), the sizes of the input and output files $S_{D,in}$ and $S_{D,out}$ (in bytes), and — in the market-based protocols — its price bid p_{Task}. It may also have a deadline t_D or a weight w_{Task}. We assume that the execution of a task can be spread over an arbitrarily large fraction of a resource. We only consider the case that there are no dependencies among the tasks. Task pre-emption is not possible, i.e. a task cannot migrate to another Server once its execution has started. Also, the task's memory requirements are not considered in our model.

Servers. There is a fixed number of Servers N_{Serv} in the system, which provide CPU resources to the Clients and charge them according to the policy of the marketplace. We assume that each Server has a resource consisting of one or more resource units (RU). This allows us to model either time-shared resources (where a resource unit corresponds to a time share of a resource) or space-shared resources (where a unit corresponds to a CPU). We further assume that a task can execute on several resource units in parallel and that each unit can be split and allocated to different tasks. To enable modelling of resources with different speed, we introduce a speed factor f_{Speed}, which is equal to 1.0 on a reference machine. On a reference machine, one resource unit will need one unit of simulation time (e.g. one second) to execute one task unit. Thus, if a constant number of resource units $N_{RU,alloc}$ is allocated to a task with size S_C, the duration of its execution is given by $\frac{S_C}{N_{RU,alloc} \cdot f_{Speed}}$.

Background Load. We also introduce *background load* on the Servers' resources, i.e. a load generated by tasks which are outside the control of the EMP. We assume a Poisson distribution for the arrivals of background tasks, each of which has the same size, $S_{C,BG}$, and number of resource units allocated, $N_{RU,BG}$. If a background task arrives at a time when no (or not enough) resource units are available, it is put into a queue. Background tasks waiting in this queue are started immediately when resources become available.

Scheduling policy. There are different ways of scheduling resource units to tasks which have been allocated by the EMP. In this paper we consider the following two cases ($N_{RU,alloc}$: number of resource units allocated, $N_{RU,avail}$: number of resource units available):

1.) allocate all available resource units of the Server to the task. The number of allocated units may increase or decrease during the task's execution due to changes in the background load (which is given priority): $N_{RU,alloc} = N_{RU,avail}$

2.) allocate a fraction of the currently available resource which is proportional to the task's price bid $p_{Task,i}$ in relation to the sum of all price bids $\sum p_{Task,i}$ on the Server (including the bid of the task itself). The allocated share may increase or decrease during the task's execution due to arrivals and departures of other tasks or changes in the background load (which is given priority): $N_{RU,alloc} = \frac{p_{Task,i}}{\sum_i p_{Task,i}} N_{RU,avail}$

Electronic Marketplace (EMP). The Electronic Marketplace (EMP) provides facilities for the Servers to advertise their resources. The parameters to be published include the number of resource units, the price per task unit and the resource's speed. For the Clients (or their agents) it provides means to search for a suitable resource and negotiate the price. Currently, the processing delays of the Clients' and the Servers' requests are neglected (until results from 'real-world' experiments are available).

Communication Model. As the actors in our systems are distributed over the Internet, communication delays need to be considered. According to experimental results in [12], the communication delay T_{Comm} on a network link i can be considered a lognormally distributed random variable. This observation is supported by [11]. In our simulation model the communication delay for a data transfer is determined by the latency and bandwidth of the network link and by the size of the transmitted data. However, for the experiments presented in this paper we make the following simplifying assumptions: we use a network topology where all actors are on different nodes, where all nodes are connected to each other, and where all network links are equal. Furthermore, all messages are the same size. We model T_{Comm} as a lognormally distributed variable with mean μ_{Comm} and standard deviation σ_{Comm}.

3 Protocols

Three different resource allocation protocols are discussed in this paper: the Round-Robin Protocol (RR), Continuous Double Auctions Protocol (CDA), and Proportional Share Protocol (PSP). CDA has been chosen, because the studied scenario requires a *double* auction, i.e. a many-to-many protocol (and not 1-to-many protocols like English or Vickrey auctions). Furthermore, it has to be a *continuous* auction where transactions are carried out immediately whenever bids or offers change. For our scenario this is preferable to using protocols where the transactions are only carried out at periodic intervals, as e.g. in the Clearinghouse auction. In such a protocol an arriving task would have to wait for the next auction — which we need to avoid in or-

der to minimise response time. The reason for examining the PSP protocol is that similar policies have been proposed for the scheduling of tasks in computational clusters [13, 14, 15]. In proportional-share scheduling, a resource is split up among several task which are allocated resource shares that are proportional to their price bids. The PSP protocol can improve on CDA for certain situations like high network latency and high resource heterogeneity. CDA and PSP are 'greedy' in the sense that a task is assigned the best possible resource that is available at a given time. We compare them to RR which is far simpler because it does not use information about load or speed of the resources for the allocation decisions. However, it is still adequate for certain situations.

First, we will describe the sequence of interactions between the actors which these protocols have in common. The different steps (1-8) are shown in Figure 1. Next, we will describe the three protocols.

Server: Registration of resources. Before any interactions at the EMP can take place, the Servers need to register their resource offers. These include the following parameters: the number of resource units available $N_{RU,avail}$, the speed factor f_{Speed} and the price per task unit p_{Serv}. The price is adjusted whenever a task (or background task) starts or completes its execution at the Server. It depends linearly on the Server's utilisation: If the Server is unloaded, it is set to the minimum (reservation) price $p_{Serv,reserv}$, whereas for full load the Server asks for the maximum price $p_{Serv,max}$. Whenever $N_{RU,avail}$ or p_{Serv} is changed, the Server updates its resource offer at the EMP.

Client: Task creation and query at the EMP. Tasks are created using an exponential distribution for the inter-arrival time τ. For each task, two objects are generated. The Task Query object contains the necessary information for a query to the EMP: the computation size S_C, the minimum price per task unit $p_{Task,min}$ (which is initially requested), the maximum price per task unit $p_{Task,max}$, the task deadline t_D, the task's ID, and a reference to its Client. The Task Data object represents the input parameters of the task. It is sent from the Client to the Server in case of a successful query. On creation of a task, the Task Query object is sent to the EMP (step 1) where it remains until an appropriate resource is found (step 2).

EMP: Process task query. Each task query which arrives at the EMP is processed immediately. If a suitable resource is available and the task's price bid is high enough, the resource is taken and the query result is sent back to the Client (step 3). The resource is considered unavailable until the task completes its execution at the Server. (An exception to this is the PSP protocol where several tasks can share a resource.) If no match is found, the task will wait at the EMP until a suitable resource becomes available (or the task's deadline has passed). For CDA and PSP a mechanism ('task price adjustment') is provided which allows Task Query objects to linearly increase their price bid p_{Task} at regular time intervals (in order to be served eventually). This aspect of the protocol, however, is not used in the simulations presented in this paper.

Server: Task execution. After receiving a query result from the EMP, the Client sends the Task Data object to the Server (step 4). The Server executes the task on the number of resource units $N_{RU,alloc}$ which have been allocated by the EMP (step 5). For Round-Robin and CDA the second resource scheduling policy is used. Therefore, the task is executed with the effective speed $N_{RU,alloc} \cdot f_{Speed}$. Note that $N_{RU,alloc}$ can vary during the task's execution. Thus, the duration of the execution is not known a priori. On completion, the resource information at the EMP is updated (step 6) and the result of the task is sent to the Client (step 7). If it arrives before the deadline, a bank transfer from the Client's to the Server's account is initiated (step 8). Otherwise, the Server is penalised and receives nothing. Note that the accounting is not relevant to the results reported in this paper.

3.1 Round-Robin Protocol

In the Round-Robin Protocol no pricing is used. The incoming task queries are matched with the 'next' available resource offer which meets the task's constraints — but which is usually not the 'best'. For this purpose an iterator is used which cycles through the list of Server resource offers.

On arrival of a Task Query object, the list of resource offers is searched until a resource is found which satisfies the task's constraints {size, price, deadline}. The search starts at the current position of the iterator. In case of success, the resource offer is taken. The result is returned and the iterator is incremented. Otherwise, the iterator is also incremented and the next resource offer is considered. This step is repeated until all resource offers have been checked or a match has been found. If the query is successful, the result is sent to the task's Client. Otherwise, the Task Query object remains at the EMP until a suitable resource becomes available. Whenever a resource becomes available, the Task Query objects at the EMP are processed (in the order of their arrival) until the resource offer is taken or all elements have been checked and no match was found.

3.2 Continuous Double Auction Protocol

The aim of the Continuous Double Auction Protocol (CDA) is to allocate the best possible resource to an arriving task and to prioritise tasks according to their price bid. When a Task Query object arrives at the EMP the protocol searches all available resource offers and returns the first occurrence of the 'best' match, i.e. the cheapest or the fastest resource which satisfies the task's constraints. Whenever a resource becomes available and there are several tasks waiting, the one with the highest price bid p_{Task} is processed first.

3.3 Proportional Share Protocol

In the Proportional Share Protocol (PSP) the second resource scheduling policy is used. The amount of resources allocated to a task depends on its price bid $p_{Task,i}$ in relation to the sum of price bids $\sum p_{Task,i}$ of all tasks executing on that Server (including the bid of the task itself). When a Task Query object arrives at the marketplace, all resource offers are checked in order to find the resource which is the fastest to execute the task and which meets the task's constraints {size, price, deadline}. The effective execution speed — which has to be maximised — depends on the background load and the sum of price bids at the Server.

Every time a task or background task starts or completes execution on the Server, the other tasks need to be rescheduled. The tasks' resource shares change, and so do their execution speeds. At such events, the effective execution speed $s_{Eff,i}$ of each task i has to be determined. First, the overall effective execution speed of the Server $s_{Eff,total}$ needs to be calculated. It is given by: $s_{Eff,total} = f_{Speed} \cdot N_{RU,avail}$. From this, the current effective execution speed of the task $s_{Eff,i}$ can be determined. It is given by: $s_{Eff,i} = s_{Eff,total} \cdot \frac{p_{Task,i}}{\sum p_{Task,i}}$.

4 Results

In this paper we aim to compare the performance of the two market-based protocols to Round-Robin. A *full* exploration of the very large parameter space is beyond the scope of this paper. Instead we focus on several examples which demonstrate the advantages of market-based resource allocation. We have chosen parameter values which are characteristic for cluster settings (homogeneous resources, negligible communication delays) and Grid settings (heterogeneous resources, significant communication delays). We use resource scheduling policies, where the resource share of arriving tasks may vary as the background load is prioritised. Such a prioritisation of the background load is motivated by a time-shared PC where the application of the local user is supposed to be unaffected by the tasks allocated by our marketplace. On a Linux machine this can be achieved by assigning a low priority to the incoming tasks — e.g. by using the `nice` command. We consider two different scenarios: one where all incoming tasks are equally important and one where tasks have different priorities.

4.1 Simulation parameters

For all simulations we use the same set of default parameters which are given in this section. In each of the following experiment one of these parameters is varied while all other parameters remain fixed. The total length of each simulation run is set to 1300.0 time units. During this time, tasks are generated by the Client(s). During the first 100.0 time units no measurements are made. This is to ensure that the system reaches a steady state. After this initial period, the number of tasks which is statistically expected to be generated during an interval of 1000.0 time units is considered in the result. To allow these tasks to complete, an additional final margin of 200.0 time units is provided. To eliminate randomness in the results, each measurement is repeated 40 times with different random seeds. For each point in our diagrams, the error bars of the 95% confidence interval of the mean are shown. For the simulated scenarios it does not matter how many Clients there are in the system. Therefore, we use only one Client for the generation of tasks. All tasks have the same computation size S_C = 10.0. Tasks of different sizes are treated equally by our protocols, therefore using different sizes would not change the result. Task deadlines are not used, and also the task input file size $S_{D,in}$ and output file size $S_{D,out}$ are set to zero. The default value of the task price bids p_{Task} is 100.0.

There are N_{Serv}= 10 Servers in the system. All Servers have the same resource size $N_{RU,total}$ = 10 and speed factor f_{Speed} = 1.0. The Servers generate background tasks, each having a computation size $S_{C,BG}$ = 10.0. Whenever a background task is started, it is allocated one resource unit at a time ($N_{RU,BG}$= 1). For the Servers a simple pricing strategy is used: the lower their utilisation, the lower the price. The price per task unit at an unloaded Server is $p_{Serv,reserv}$=0.0. At full load it is $p_{Serv,max}$=100.0. The inter-arrival time of tasks at the Client τ is such that the average load of tasks in the system is 40% of the total Server capacity. Similarly, the inter-arrival time of background tasks at the Servers is

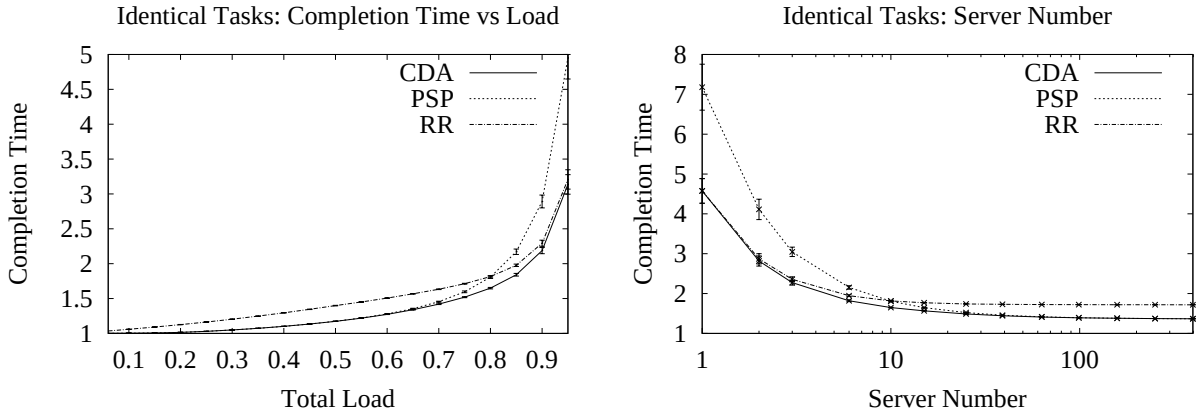

Figure 2: From left: Variation of the load and the server number (identical task scenario)

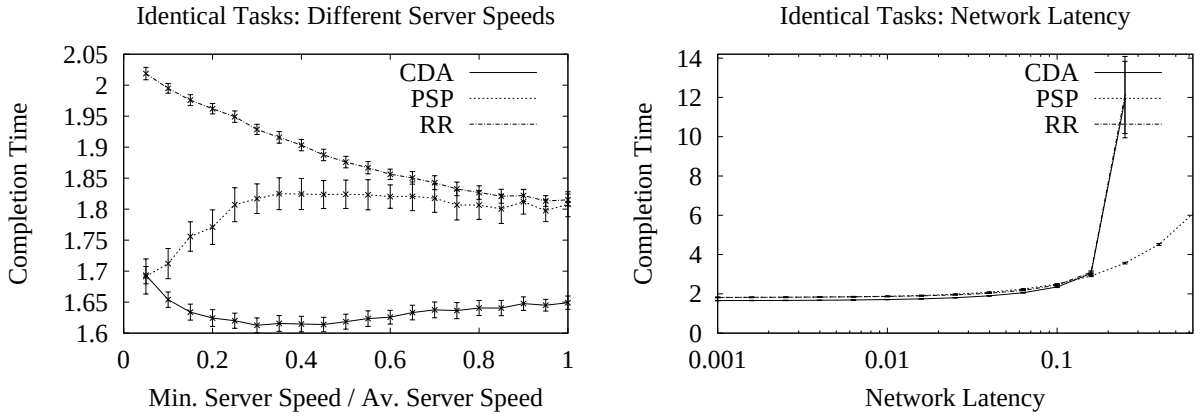

Figure 3: From left: Variation of the server speeds and the network latency (identical task scenario)

set to a value which results in an average background load of 40% of the capacity of each Server. This leads to an average total load of 80% in the system. Except for one experiment the communication delay T_{Comm} is set to zero. In our experiments the following parameters will be varied: 1st, the overall load in the system, 2nd, the number of Servers, 3rd, the resource heterogeneity, and 4th, the network latency.

4.2 Experiments: Identical task scenario

In the first set of experiments all tasks have the same price bid. The mean completion time of the tasks is the metric to be minimised by the protocols. This metric can be considered the utility function of the Client: the lower the metric, the higher the utility.

Variation of the load. In Figure 2 (left) the mean completion time for the different protocols is shown when the total load — half of which is background load — is varied between 0 and 90% of the total resource capacity in the system. CDA provides the best results for the whole range of values. Round-Robin performs worse because resources are allocated arbitrarily, whereas CDA selects the cheapest available resource for a task (which in this case is the least loaded and therefore the fastest). The difference is highest at a load of about 50%, where Round-Robin's completion time is 20% higher than CDA's. PSP performs almost as well as CDA as long as the load is low. The reason is that it also selects the fastest available resources for the tasks. However, for a high load (90%) its mean completion time is 32% higher. PSP allocates arriving tasks to Servers even when they are busy. Thus, it delays tasks which are already executing.

Variation of the server number. In Figure 2 (right) the number of Servers N_{Serv} in the system is varied while the load and background load ratios are kept constant. For all protocols the mean of the completion time goes down as N_{Serv} is increased. The reason is that with in-

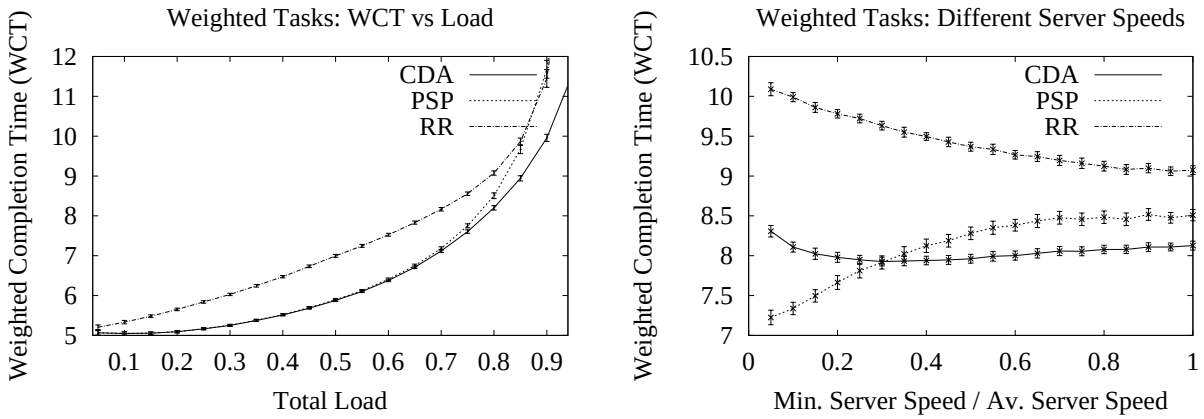

Figure 4: From left: Variation of the load and the server speeds (weighted task scenario)

creased size of the market a shortage of resources is less likely, because the overall amount of resources offered is more stable. CDA performs best for the observed range of values. For a high server number, PSP approaches its performance, because it becomes less likely that several tasks execute on a resource at the same time. Round-Robin performs worse: For N_{Serv}=398 its completion time is 26% higher which can be explained by its indifferent allocation of resources.

Variation of the resource heterogeneity. Next, we examine how the protocols cope with different degrees of heterogeneity among the Server resources (see Figure 3 (left)). The speed factors of the Servers are evenly distributed between a minimum $f_{Speed,min}$ and a maximum 2.0 - $f_{Speed,min}$. This choice of the maximum value ensures that the overall resource capacity in the system is constant and that the average speed factor is 1.0. In the experiment the parameter $f_{Speed,min}$ is varied within the range of 0.0 to 1.0. For CDA, tasks select the fastest resources, since selecting the cheapest would result in lower performance. The performance of Round-Robin degrades as heterogeneity is increased. This is because it does not consider speed and load for the choice of resources. For $f_{Speed,min} = 0.05$ the mean completion time is 11% higher than for identical resources. CDA's degradation is only about 2.5%. For PSP the performance even improves: The mean completion time is 6% lower than for identical resources. The reason is that tasks can choose faster resources and are therefore more likely to execute at a high speed.

Variation of the network latency. So far, the communication delay between Clients, Servers and the EMP has been neglected. Figure 3 (right) shows the performance of the different protocols when the mean μ_{Comm} of the communication delay T_{Comm} is varied. The standard deviation of T_{Comm} is set to 50% of μ_{Comm}. A sharp rise of the mean completion time can be observed for Round-Robin and for CDA when the latency is increased over 0.15. This is because resources which are released, need to be advertised at the EMP before they can be used again. They remain idle during the communication delays, leading to a shortage of available resources in the system. For PSP, however, the rise is much slower. The reason is that the tasks are allocated immediately after arrival without waiting for Servers to become available. The resources are used during the communication delay, thus avoiding the shortage.

4.3 Experiments: Weighted task scenario

In the weighted task scenario it is examined how the protocols perform when the generated tasks have different priorities. Tasks are assigned weights w_{Task} for which a uniform distribution between 0.0 and 10.0 is used. The task price bids are set to $p_{Task} = 100 \cdot w_{Task}$. In CDA, the tasks select the resources that allow the fastest execution. As a metric the mean weighted completion time (WCT) is used, which is the mean of the task weights times their completion times. Again, this metric corresponds to the utility function of the Client.

Variation of the load. Figure 4 (left) shows the mean weighted completion time for the tasks with priorities when the total load in the system is varied. CDA and PSP perform almost equally well when the load is low and are both better than Round-Robin. As the load is increased, CDA performs best and the difference to the other protocols becomes larger: at 90% load, the mean WCT of Round-Robin is 15%, and that of PSP 17% higher than CDA's. When the load is increased even further, the gap becomes even wider which is due to the prioritisation of tasks. For the same reason PSP now

performs about as well as Round-Robin when the load is high (95%).

Variation of the resource heterogeneity. In Figure 4 (right) the server speed is varied in the same way as in Figure 3 (left). As before, Round-Robin degrades with increased heterogeneity: at $f_{Speed,min} = 0.05$ the WCT is 11 % higher than for identical resources. CDA degrades slightly, whereas PSP improves by about 15% and performs even better than CDA. Overall, the market protocols can cope better with resource heterogeneity than Round-Robin. The gain is larger than in the identical task scenario which is due to the prioritisation of tasks.

5 Conclusion

It is often assumed that market-mechanisms are better than conventional approaches. We developed a simulation model where we compared the performance of two market-based resource allocation protocols to a round-robin approach. We explored several scenarios and presented some examples in this paper.

To summarise, for a cluster of homogeneous resources — which is typical for a lab of Linux machines — the Continuous Double Auction Protocol (CDA) will perform best. However, if the load is low, the differences between the three protocols are small, and using the computationally less expensive Round-Robin protocol might be sufficient. For a situation where there is a choice of resources with different quality or load — as it is the case in a computational Grid — the results of Round-Robin will be worse than for the two market-based protocols. This is due to their allocation of the fastest possible resources and the prioritisation of tasks with high weights. The Continuous Double Auction Protocol (CDA) will perform best in most cases. However, for a high number of resources the performance of the Proportional Share Protocol (PSP) will be comparable. PSP also benefits from a scenario with very high resource heterogeneity where it may even outperform CDA. It can also better cope with situations where the communication delays are large in comparison to the computational size of a task — which is typical for Grid settings. On the negative side, its concurrent execution of several tasks at a Server results in longer individual completion times than sequential execution. This leads to poor performance if the load is very high.

These results are only examples and they are also limited to independent tasks. As future work, we will consider scenarios where the tasks have dependencies, deadlines, or where pre-emption is used. Also, the performance of our market protocols will be compared to other conventional strategies which perform better than Round-Robin. Furthermore, we plan to carry out experiments in a real Grid-like environment in order to verify our simulation model.

References

[1] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic models for allocating resources in computer systems. In *Market-Based Control: A Paradigm for Distributed Resources Allocation*. World Scientific, 1996.

[2] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Trans. Software Engineering*, 18(2):103–117, 1992.

[3] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the Internet - the POPCORN project. In *Proc. of the 18th Int. Conf. on Distributed Computing Systems*, Amsterdam, Netherlands, 1998. IEEE.

[4] M. Backschat, A. Pfaffinger, and C. Zenger. Economic-based dynamic load distribution in large workstation network. In *Proc. of the 2nd Int. Euro-Par Conf.*, volume 2, pages 631–634, Lyon, France, 1996. Springer.

[5] R. Buyya and S. Vazhkudai. Compute Power Market: Towards a Market-Oriented Grid. In *Proc. of the 1st Int. Conf. on Cluster Computing and the Grid, CCGrid'01*. IEEE, 2001.

[6] D. Abramson, R. Buyya, and J. Giddy. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computer Systems (FGCS) Journal*, 18(8):1061–1074, October 2002.

[7] J. Bredin, D. Kotz, and D. Rus. Market-based resource control for mobile agents. In *Proc. of the 2nd Int. Conf. on Autonomous Agents*, Mineapolis, USA, 1998. ACM Press.

[8] A. Chavez, A. Moukas, and P. Maes. Challenger: A multi-agent system for distributed resource allocation. In *Proc. of the 1st Int. Conf. on Autonomous Agents*, Marina del Ray, CA, USA, 1997. ACM Press.

[9] Tuomas Sandholm. Distributed rational decision making. In Gerhard Weiss, editor, *Multi-agent systems*. MIT Press, 2000.

[10] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load-balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, 1997.

[11] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Trans. Networking*, 9(4):392–403, 2001.

[12] M. Schroeder and L. Boro. Does the restart method work? Preliminary Results on Efficiency Improvements for Interactions of Web-agents. In *Proc. of the Workshop on Infrastructure for MAS at Agents01*, 2001.

[13] A. Messer and T. Wilkinson. Power to the process. In *Workshop on Parallel, Emergent, and Distributed Computing*, Reading,UK, May 1996. MIT Press.

[14] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: An economy driven job scheduling system for clusters. In *Proc. of the 6th Intl. Conf. on High Performance Computing in Asia-Pacific Region (HPC Asia 2002)*, Bangalore, India, December 2002.

[15] C. A. Waldspurger. *Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, MA, USA, 1995.