Agent-based Grid Scheduling with Calana

Mathias Dalheimer[1], Franz-Josef Pfreundt[1], and Peter Merz[2]

[1] Fraunhofer Institut für Techno-und Wirtschaftsmathematik, Kaiserslautern, Germany
{dalheimer|pfreundt}@itwm.fhg.de
[2] Department of Computer Science, University Kaiserslautern, Germany
peter.merz@ieee.org

Abstract. Grid resource allocation is a complex task that is usually solved by systems relying on a centralized information system. In order to create a lightweight scheduling system, we investigated the potential of auctions for resource allocation. Each resource provider runs an agent bidding on the execution of software with respect to local restrictions. This way, the information system becomes obsolete. In addition, each provider can implement different bidding strategies in order to reflect his preferences.

1 Introduction

Grid Computing as a way of virtualizing computational resources is becoming more relevant in research and industry [1]. It can be foreseen that grid computing will become important in a commercial scenario: service providers will sell computational power and storage. Users will buy the required amount of processor cycles and disk capacity on a per-use basis.

In the *Fraunhofer Resource Grid* (http://www.fhrg.fhg.de), we have a strong bias towards commercial grid applications. Our goal is to provide users from both industry and research with a stable, general purpose grid infrastructure. As a part of this effort, we developed a prototype of Calana, an agent-based grid scheduler.

The paper is organized as follows: section 2 provides an overview of the related work. In section 3, we introduce the architecture and design principles of Calana. Section 4 discusses the results from experiments which have been done to evaluate the system. Section 5 concludes with discussing the advantages of the architecture.

2 Related Work

The problem of grid scheduling is often referred to as *resource management*. In addition to the five challenges of resource management [2], another challenge arises from the different *stakeholder's objectives*: a grid user wants to calculate fast and cheap, but a resource provider's interest is to earn money or utilize the unused computing power of his own resources [3][4].

Currently, many projects include the development of a grid scheduler. Globus Toolkit 4.0 integrates the Community Scheduler Framework (CSF) created by Platform Computing [5]. Other schedulers include Condor [6] and Gridbus [7], which is the successor of Nimrod-G [8]. However, the projects mentioned do not tackle two very important issues: Currently, resource information is pushed periodically in information systems like the Globus Meta Directory Service [9]. During scheduling, potentially outdated information is used by the scheduler. Furthermore, stakeholders are not able to express complex preferences and adjust their strategies quickly, which is a strong prerequisite in order to establish a computational economy [10]. In real economies, these problems are solved by the established market structures [11].

Ernemann and Yahyapour describe an architecture based on economic principles in order to solve the problem of different stakeholders' objectives [12]. They use objective functions in order to find an equilibrium of interests. Although it should be possible to implement complex strategies within this system in general, they do not discuss it. Furthermore, this system allows various forms of collusion which may not be tolerated.

Various authors have discussed the use of auctions for resource allocation problems [13][11][12]. Although Wolski and colleagues [14] prefer commodities markets over auctions, the common understanding of most authors is that auctions are capable of solving a multicriteria resource allocation problem. The setup of Wolski et al. is not applicable to this work: They separated processor and disk allocation, reducing the auction's reliability to allocate all needed resources.

Taking the stakeholder's objective problem into account, we believe that economic scheduling provides a suitable solution to the grid scheduling problem. As we show in the remainder of the paper, we can incorporate different preference structures and eliminate the need for a centralized information system for dynamic information.

3 The Architecture of Calana

The design goal of Calana is to provide a lightweight and flexible scheduling system. Basically, the architecture consists of two software components: The broker and the agent. Each resource runs a small software agent. The agent registers itself to the broker. When a job needs to be scheduled, the broker uses an auction to determine which resource is available. The registered agents receive a request to bid on the application execution. They need to check the feasibility of the request:

1. The auction announcement includes a link to the software's description. The agent may now compare the application's prerequisites with its resources specification. If all prerequisites are fulfilled, the job can be further considered by the agent.
2. Then, the agent checks the local resource usage directly by trying to get an advance reservation for the job. The local resource system usually optimizes the reservation details, e.g. by using a backfilling algorithm [15]. The result are fixed start- and end-times for the job.
3. The reservation can be used to create a bid in the auction. The process of bid creation is influenced by the provider's strategy.

After the auction ends, the broker determines which bid fulfills the user's requirements best. Each resource provider can implement own agents, enabling them to specify all kinds of strategy and integrate various local resources.

In the following sections, we will describe the scheduling process in detail.

3.1 Using Auctions for Resource Allocation

Auctions can be classified as multilateral negotiations [16]. An auction is always based on a well-defined scheme: During the bidding phase, all *bidders* submit bids to the *auctioneer*. During the transaction phase, the auctioneer uses the *scoring rule* to evaluate all bids and select the best. This makes the implementation of a software system for auctions easy [11].

There are many different auction types. Each one has certain rules for both bidding and transaction phase and a certain impact on the fairness of the market. For the prototype implementation, we use a first-price sealed-bid auction [17]: the bids are not visible to other bidders. The best bid wins the auction, and the price

of the auction is also determined by the best bid. Another possible auction is the *vickrey auction* which is known to deliver pareto-optimal allocations as well as it prevents some forms of collusion [17][16][18].

In order to consider all stakeholder's preferences, resource selection must implement a *multi-criteria optimization* as defined by Kurowski et al [19]. We use multicriteria bids: a bid may contain multiple values [16]. In general, all auctions can be adopted to work with multicriteria bids: An announcement of valid properties of a bid has to be made. For the comparison of bids, the auctioneer can use a scoring rule that delivers a relative value of a bid. The bid with best relative value will win the auction.

A serious problem of auctions is collusion: a bidder may try to break auction rules in order to increase his benefit. Although a bidder may only submit bids for himself, there are ways to influence the market [18]: Pools of bidders may influence auctions, the auctioneer may trade with information in sealed-bid auctions and offerors can use phantom bids to increase the market's competition. Creating fair auctions is a complex task. Basically, the use of sealed bid auctions in combination with a trusted auctioneer seems to be reasonable [16]. Further research concerning auction design for computational markets is needed.

3.2 The Calana Prototype

An overview is given in figure 1. The broker works as an auctioneer for scheduling requests, while the agent submits bids for the auctions. The agents are located at the provider's sites, while the broker must be hosted by a trusted third party. The user accesses the broker not directly but by using a portal or a workflow tool, e.g. the GridJobHandler [20].

For each job, the user's tool calls the broker to retrieve a schedule. Each call consists of a description of the software, possibly the size of the datasets and some weights for the scoring function in order to reflect the user's preferences. One may also consider scoring functions completely specified by the user.

The broker receives the request and creates a new auction. A *call for bids* is sent to each registered agent. This call contains a software description, provided by the manufacturer of the software, and the auction deadline. The software description contains a description of the runtime behaviour, along with other properties such as the number of cluster nodes the software should run on. The agent can use this information to predict the overall runtime of the application. Of course, this prediction is not reliable, only for certain types of software the runtime may be calculated, e.g. depending on the input data [21][22]. As a fallback, user-specified walltimes can be used.

Based on this information, the agent composes a bid. This can be influenced by the local bidding strategy: Jobs may be cheaper on weekends or promotions during holidays may be considered. A bid consists of the estimated job finish time t^{f} and a monetary price p for the execution. When a bid is submitted, an advance reservation for the software must be made in order to be able to satisfy an auction won.

When the auction deadline has passed, the broker judges all bids by applying a scoring rule. This enables the broker to consider user preferences modeled as weights for the scoring rule. For the value v of a given bid, the broker calculates the scoring rule with the parameters given by the users. For example, the value of the bid i, $0 \leq i \leq n$, may be calculated as follows:

$$v_i = f(p_i, t_i^{\mathrm{f}}) = g \cdot \frac{p_i}{p_{max}} + (1 - g) \cdot \frac{t_i^{\mathrm{f}}}{t_{max}^{\mathrm{f}}} \tag{1}$$

with a weight g for the price preference, $0 \leq g \leq 1 \in \mathbb{R}$, maximum finish time $t_{max}^{\mathrm{f}} = \max\{t_i^{\mathrm{f}} | 0 \leq i \leq n\}$ and maximum price $p_{max} = \max\{p_i | 0 \leq i \leq n\}$. The

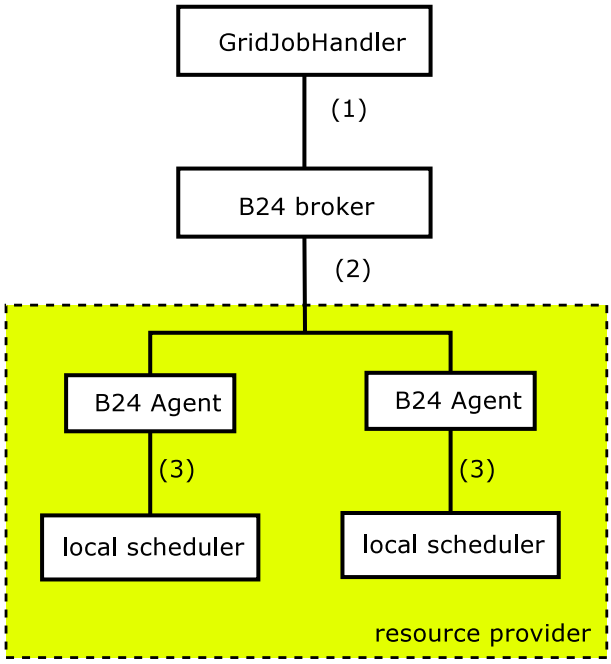


Fig. 1. Overview of the architecture: the user workflow tool uses the broker component to create a valid schedule (1). Resource providers use an agent in order to connect their local scheduling systems to the architecture (3). The broker and agents interact to create the schedules (2).

best bid b is the bid with the minimal value:

$$b = \min\{v_i | 0 \leq i \leq n\} \tag{2}$$

This way, a user has the opportunity to express even fuzzy personal preferences, like "I want my results fast, but I don't want to pay that much." In this example, one may choose $g = 0.2$. The scoring rule can be generalized in order to consider more variables or to respect user thresholds like "I prefer fast execution, but it should not cost more than 11.2€".

Finally, after a scheduling decision has been made, the broker notifies the user's tool, which may use the advance reservation to execute the job. In parallel, all agents that haven't succeeded will be notified to cancel the advance reservation. If a notification is not received, the agents drop reservation after a timeout. Since all transactions are made by a central broker, an accounting service may also be provided.

The providers register themselves at the broker. There must be an announcement of available brokers, but since their number is comparatively low, this should not have negative effects. The auction announcement is broadcasted to all registered agents. Only the interested agents answer, and only these will receive a message about the auction result. Agents may also choose to resend a bid, e.g. if they didn't win an auction at another broker. When a sealed auction is used, the current bid must not be broadcasted to all agents. If n agents are attached, we expect the number of messages to be propotional to n. Compared to a system with a central information system, more messages are needed during scheduling.

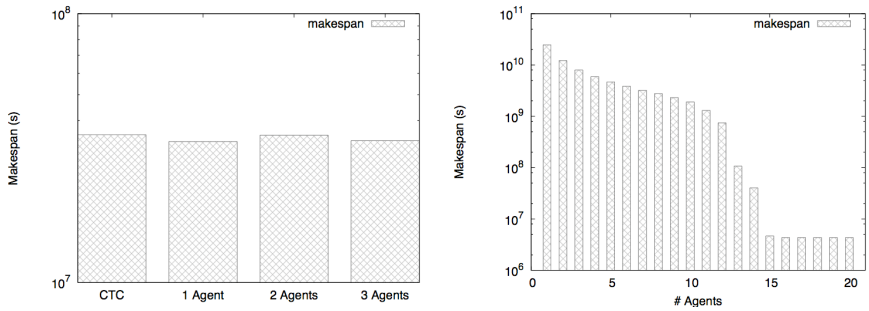
But when comparing to other schedulers, the periodical messages to update the dynamic information in centralized information system are often not taken into

account. In this architecture, no centralized dynamic information is needed: since the agents reside beneath the resources, they can directly access the batch queues, getting up-to-date information.

The amount of messages can be reduced further by the introduction of a tree-like hub structure: The broker announces all auctions to its hub peers, which have the resources attached. They propagate the auction request to the resources and collect the bids. Finally, results are handed back to the broker. This infrastructure can also be used to reduce information: for example, a hub peer might choose only to propagate the pareto-dominant bids and drop the rest. This way, the messages of the broker are proportional to $\log n$. Note that all hubs must belong to a trustworthy institution in order to avoid collusion.

4 Experiments

Unfortunately, there is no common benchmark for grid schedulers so far. Lacking a common model for workload creation [23], we used parallel workload sets from the Cornell Theory Center (CTC)[24] and created our own job sets for non-parallel workloads. A model for grid resources has been proposed by Kee and colleagues [25]. In future work, we will use this information to define and run further experiments. In addition to our prototype, we developed an event-based simulation with Gridsim [26] which we use to run most experiments. We denote if we used the prototype. We use two measurements for the evaluation: The total completion time and the



(a) Total completion time comparison (b) Scalability for nonparallel jobs

Fig. 2. The left figure shows a comparison of the total completion time for the CTC scheduler and Calana, on the right side results of our non-parallel workload test are shown.

overall price. The total completion time C provides the time needed to run n jobs in total, including queuetime and runtime of each individual job:

$$C = \sum_{i=1}^{n} (queuetime_i + runtime_i) \tag{3}$$

The overall price is the sum of the prices of all single software executions.

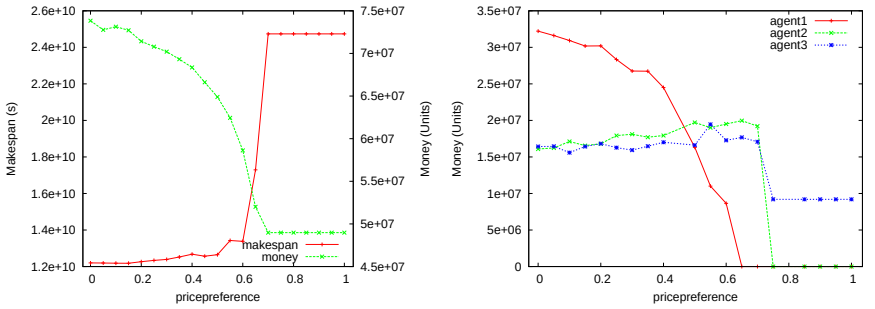
4.1 Comparison to other schedulers

The lack of a common benchmark makes the comparison of schedulers difficult. Since the CTC workload is freely available and delivers the decisions of a centralized scheduler based on backfilling, we can directly compare our parallel simulation runs,

see figure 2a: We compared the allocation of the CTC scheduler with three runs of the prototype. In the first run, one agent was responsible for a cluster as big as the 420 node CTC cluster. In the second and third run, we added agents that managed one (114 node) resp. two smaller (56 node) clusters in addition to a cluster with 306 nodes managed by the first agent. This is necessary because the biggest job in the CTC's workload needs 306 nodes. As the figure shows, the runtimes are almost the same, independent of the number of attached agents. Therefore, we do not expect centralized scheduling systems to perform better in general.

4.2 Scalability of the architecture

Our prototype was able to handle 500 concurrent agents, bidding on parallel jobs. Since the EGEE project has about 150 sites, we would be able to schedule more than three times the EGEE project with a single broker. Lacking a model to create appropriate workload, we used the CTC workload as input. Of course, the workload is too low to utilize this number of sites, so the queuetime was zero for all runs. For the following experiments, we used a synthetic workload which contains nonparallel jobs. We simulated successively 1 to 20 agents, see figure 2b. Since each agent manages a non-parallel resource, the total completion time for the workload decreases continuously until all jobs can be processed instantly.



(a) Influence of user preferences (b) Influence of exponential smoothing

Fig. 3. Influence of user preferences and bidding strategies on the allocation. When the price component of the bids become more important, the behaviour of Calana changes.

4.3 Influence of user preferences and provider strategies

In our basic model, we assume each job has a fixed setup price p^s and a constant price p^t for each CPU-second. The price p of a computation running f^{t} seconds can be calculated as $p = p^s + f^{\mathrm{t}} \cdot p^t$. A provider may now choose different values of p^s and p^t. At the same time, the user wants to specify his own preferences. In order to show Calana's ability to deal with this, we ran an experiment with two agents.

The first agent uses a setup price p_1^s which is half of the setup price of the second agent ($2p_1^s = p_2^s$), but the computation cost is more expensive: the price per CPU-second p_1^t is twice times of the second ($p_1^t = 2p_2^t$). As figure 3a shows, the overall price drops while the total completion time increases when the user's preference changes to cheaper execution. When the price preference reaches 70 % ($g \geq 0.7$), the values doesn't change any more: The price dominates the broker's decisions.

Of course, this experiment raises the question how a resource provider can adjust its agent to the market. We used the settings above to evaluate a third agent that

generates p_3^t based on the last auctions: It uses exponential smoothing ($\alpha = 0.15$) to wheight the last 10 winning bid's amount $\hat{p}$ and uses this value to create a bid:

$$p_3^t = \alpha \sum_{i=0}^{10} (1-\alpha)^i \cdot \hat{p}_i \tag{4}$$

As shown in figure 3b, the third agent dominates the scenario completely if the price influence becomes important.

5 Summary

In this paper, we presented an auction-based grid scheduler that is capable of taking both user and resource provider preferences into account. No central information system for dynamic data is necessary, providing a major advantage compared to other schedulers. A provider may implement all kind of strategies and change them frequently. By creating a bid, it is possible to reflect local restrictions. User preferences can be taken into account by using multicriteria bids.

The architecture is scalable, portable and extensible. In general, the software can be adopted to fit other environments. The architecture will be developed further so that it will be usable with other grid setups. In future, the architecture will be evaluated in a real industry setup and enhanced to include more market aspects. An evaluation of different agent strategies will be made, along with a production-ready implementation in the Fraunhofer Resource Grid.

Acknowledgement

The Competence Center High Performance Computing at the Fraunhofer ITWM (`http://www.itwm.fhg.de`) is supporting this work.

References

1. Foster, I., Kesselman, C.: Computational Grids. In: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers (1998)
2. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S.: A resource management architecture for metacomputing systems. In: Job Scheduling Strategies for Parallel Processing. Volume 1459 of LNCS., Springer (1998) 62–68
3. Moreno, R., Alonso-Conde, A.B.: Job scheduling and resource management techniques in economic grid environments. In et al., F.F.R., ed.: Across Grids 2003. Volume 2970 of LNCS., Springer (2004) 25–32
4. Ernemann, C., Hamscher, V., Yahyapour, R.: Economic scheduling in grid computing. In: Job Scheduling Strategies for Parallel Processing. Volume 2537 of LNCS., Springer (2002) 128–152
5. Smith, C.: Open source metascheduling for virtual organizations with the community scheduler framework (csf). Technical report, Platform Computing, Inc. (2003)
6. Roy, A., Livny, M.: Condor and preemptive resume scheduling. In Nabrzyski, J., Schopf, J.M., Weglarz, J., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
7. Venugopal, S., Buyya, R., Winton, L.: A grid service broker for scheduling distributed data-oriented applications on global grids. Technical Report GRIDS-TR-2004-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia (2004)
8. Abramson, D., Buyya, R., Giddy, J.: A computational economy for grid computing and its implementation in the Nimrod-G resource broker. Future Generation Computer Systems **18** (2002) 1061–1074

9. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. In: Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press (2001)
10. Kenyon, C., Cheliotis, G.: Grid resource commercialization. In Nabrzyski, J., Schopf, J.M., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
11. Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic models for resource management and scheduling in grid computing. The Journal of Concurrency and Computation: Practice and Experience (CCPE) **14** (2002) 1507–1542
12. Ernemann, C., Yahyapour, R.: Applying economic scheduling methods to grid environments. In Nabrzyski, J., Schopf, J.M., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
13. Wellman, M.P., Walsh, W.E., Wurmann, P.R., MacKie-Mason, J.K.: Auction protocols for decentralized scheduling. In: 18th International Conference on Distributed Computing Systems, Amsterdam. (1999) Revised and extended version of "Some economics of market-based distributed scheduling".
14. Wolski, R., Plank, J.S., Brevik, J., Bryan, T.: Analyzing market-based resource allocation strategies for the computational grid. International Journal of High Performance Computing Applications **15** (2001) 258–281
15. Lifka, D.A.: The ANL/IBM SP scheduling system. In Feitelson, D.G., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Number 949 in LNCS, Springer (1995) 295–303
16. Peters, R.: Elektronische Märkte - Spieltheoretische Konzeption und agentenorientierte Realisierung. Physica Verlag (2002)
17. Schimmel, K., Zelewski, S.: Untersuchung alternativer auktionsformen hinsichtlich ihrer eignung zur koordination verteilter agenten auf elektronischen märkten. Technical Report 19, Institut für Produktionswirtschaft und industrielle Informationswirtschaft, Universität Leipzig (1996)
18. Corsten, Hans; Gössinger, R.: Auktionen zur marktlichen koordination in unternehmungsnetzwerken. In Corsten, H., ed.: Unternehmungsnetzwerke. Oldenbourg (2001)
19. Kurowski, K., Nabryski, J., Oleksiak, A., Weglarz, J.: Multicriteria aspects of grid resource management. In Nabrzyski, J., Schopf, J.M., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
20. Hoheisel, A.: User tools and languages for graph-based grid workflows, 2004. In: Grid Workflow 2004 Special Issue of Concurrency and Computation: Practice and Experience. (2004)
21. Merten, D.: Integrated performance analysis of distributed computer systems. Workshop on Performance Characterization, Modeling and Benchmarking for Existing and Emerging HPC Systems (2004)
22. Russell, M., Allen, G., Goodale, T., Nabrzyski, J., Seidel, E.: Application requirements for resource brokering in a grid environment. In Nabrzyski, J., Schopf, J.M., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
23. Chapin, S., Cirne, W., Feitelson, D., Jones, J., Leutenegger, S., Schwiegelshohn, U., Smith, W., Talby, D.: Benchmarks and standards for the evaluation of parallel job schedulers. In Feitelson, D., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Volume 1659 of LNCS., Springer (1999) 66–89
24. Feitelson, D.: Parallel workloads archive (2005) `http://www.cs.huji.ac.il/labs/parallel/workload`.
25. Kee, Y.S., Casanova, H., Chien, A.A.: Realistic modeling and synthesis of resources for computational grids. In: Proceedings Supercomputing 2004, IEEE (2004)
26. Buyya, R., Murshed, M.: Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. The Journal of Concurrency and Computation: Practice and Experience (CCPE) **14** (2002)