International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:10, 2007

Grid Coordination with Marketmaker Agents

Xin Bai, Kresimir Sivoncik, Damla Turgut and Ladislau Bölöni

Abstract**—** Market based models are frequently used in the resource allocation on the computational grid. However, as the size of the grid grows, it becomes difficult for the customer to negotiate directly with all the providers. Middle agents are introduced to mediate between the providers and customers and facilitate the resource allocation process. The most frequently deployed middle agents are the matchmakers and the brokers. The matchmaking agent finds possible candidate providers who can satisfy the requirements of the consumers, after which the customer directly negotiates with the candidates. The broker agents are mediating the negotiation with the providers in real time.

In this paper we present a new type of middle agent, the marketmaker. Its operation is based on two parallel operations - through the *investment process* the marketmaker is acquiring resources and resource reservations in large quantities, while through the *resale process* it sells them to the customers. The operation of the marketmaker is based on the fact that through its global view of the grid it can perform a more efficient resource allocation than the one possible in one-to-one negotiations between the customers and providers.

We present the operation and algorithms governing the operation of the marketmaker agent, contrasting it with the matchmaker and broker agents. Through a series of simulations in the task oriented domain we compare the operation of the three agents types. We find that the use of marketmaker agent leads to a better performance in the allocation of large tasks and a significant reduction of the messaging overhead.

Keywords**—** grid computing, autonomous agents, market-based grid

I. Introduction

The purpose of the computational grid is to make the resources of the providers available to the consumers. This leads to a complex control and coordination problem, which is frequently solved with a market model based on real or virtual currency. To support a market based grid, we need a collection of middleware services: directories, security and authentication, accounting and banking and so on. By dynamic pricing of the resources, negotiation and auction processes we hope to achieve an efficient and fair distribution of resources. For a small grid it is possible for a client to perform negotiations with all the possible providers in order to select the one which best satisfies its requirements. However, as the size of the grid grows, it becomes difficult for the customers to negotiate directly with all the providers. To make these negotiations more efficient, a new set of web components need to be introduced, whose goal is to mediate the interaction between the customers and providers. As these components have considerable autonomy and frequently need to take initiative, they are best viewed as software agents. In the light of their role as mediators, we will call them middle-agents [14].

The most frequently employed middle-agents in market oriented grid architectures are the matchmaker and broker agents. The matchmaking agent finds possible candidate providers who can satisfy the requirements of the consumers, after which the customer directly negotiates with the candidates. The broker agents are mediating the negotiation with the providers in real time.

In this paper, we propose a new type of middle agent, the marketmaker, which decouples the allocation of the resources from the execution of the tasks, by buying larger chunks of resources from the providers and reselling them to the consumer agents. Our simulation studies show that the presence of the marketmakers contributes to a more efficient resource distribution, and provides efficient levers for the control of the grid environment.

The remainder of this paper is organized as follows. In Section II we introduce the marketmaker agents, discuss the design space of its specific investment and resale strategies, and present the resource allocation and communication models considered in this paper. The setup of experimental scenarios and the results of the experiments are presented in Section III. In Section IV we review previous work done on market based grid architectures, emphasizing the different implementations of middle agents such as brokers and matchmakers. We conclude in Section V.

II. Grid coordination with marketmaker agents

A. The operation and strategies of marketmaker agents

The role of the middle agents in the market grid is to mediate between the clients and the providers. In this section, we introduce the marketmaker agent and contrast its functionality with the matchmaker and broker agents.

The matchmaker agent maintains a knowledgebase of the resources. It receives a requirements description from the clients and returns to the client one or more *matches*. It is not involved in the actual resource allocation process. It is paid according to a scheme which depends on the number and/or quality of matches. The main role of the matchmaker is to exploit economies of scale in the querying of available resources, by acquiring information from the clients and reselling it to interested parties.

A broker agent receives a requirement description from the client and through a three way negotiation process participates in the allocation of the resources. The broker might use the services of other agents such as matchmakers or marketmakers. In the interaction with the client, the broker provides a

Manuscript received November 15, 2005
Xin Bai is with the University of Central Florida (email: xbai@cs.ucf.edu).
Kresimir Sivoncik is with the University of Central Florida (email:sivi@hector.cs.ucf.edu)
Damla Turgut is with the University of Central Florida (email:turgut@cs.ucf.edu)
Ladislau Bölöni is with the University of Central Florida (email:lboloni@cs.ucf.edu)

series of *offers* which include resource allocation and price, which the user might accept or reject. The broker agent is paid by the client and/or the provider, according to a pricing rule which depends on the success of the transaction, the size of the transaction, and the level of satisfaction of the user or the provider. In general, the interest of the broker agents is to maximize the number of successful resource allocations.

A marketmaker agent performs similar functionality with the broker from the point of view of the client. From the point of providers, however, the marketmaker can perform advance allocation of the resources either through *buying* or *optioning* of the resources. The marketmaker agent buys and sells resources, thus acts as a "superprovider" which aggregates resource reservations. The high-level operation of the marketmaker is described in the following algorithm.

Algorithm 1 Marketmaker meta-algorithm

When available money > trigger
(1) select providers $\{P_{i1}, P_{i2} \dots P_{in}\}$
(2) negotiate usage chunks $\{c_{i1}, c_{i2} \dots c_{in}\}$
(3) purchase most advantageous chunk $c_{i,best}$
When request from client C for task T with resource requirements R arrives
verify if request is satisfiable
(4) generate a set of choices
(5) negotiate price based on these choices
(6) allocate task T according to allocation policy

The steps numbered (1-6) represent specific strategies which needs to be decided at the implementation of the marketmaker agent. The steps (1-3) collectively represent the *investment strategy* of the agent, while steps (4-6) are the *reselling strategy*. The choices for these strategies provide a very large design space for the marketmaker agent.

(1) The selection of the negotiation set, the set of providers with which the marketmaker negotiates simultaneously. In general, the larger the negotiation set, the more advantageous deals can be achieved, but a large negotiation set leads to large negotiation overhead.

(2) The negotiation of the reservation chunks and their prices is the most complex part of the investment strategy. The chunks can be suggested both by the marketmaker and the provider. The marketmaker agent needs to have an *investment target*, that is the amount and temporal distribution of resources it is trying to acquire. The agent might deploy various external information, historical data and future load predictions to determine its investment target.

(3) The purchase decision is made by evaluating the offers according to their prices and their perceived value depending on the investment target and existing reservations.

(4) This step determines the ways in which the marketmaker can satisfy the request from its existing reservations. These choices are not necessarily communicated to the client, but they are used as the basis of negotiation.

(5) The negotiation of the price between the marketmaker of the provider can follow one of the well-known negotiation algorithms, such as the monotonic concession protocol with the Zeuthen [22] strategy. The parallel negotiations with multiple clients can modify the perceived value of the resource for the marketmaker.

(6) Once the marketmaker agent accepted to satisfy the request it needs to find the most efficient way to satisfy the request, such that it maximizes its ability to satisfy future requests. At minimum, this is equivalent with a bin packing problem, but it can also include a probabilistic evaluation of the success of the current negotiations as well as predictions of future requests.

Let us consider how the selfish interest of the marketmaker can benefit the grid as a whole. The profit of a marketmaker is the difference between the selling and the buying prices. For the marketmaker agent, for instance, once it bought a certain amount of future resources, its best interest is the efficient allocation of those resources. On the other hand, if certain resources are not bought up by the marketmakers, it is a signal that the grid as a whole is overweighted in terms of resources. This can provide a useful input for the managers of the grid.

The fundamental advantage of a marketmaker over matchmaker and broker agents is that its selfish interests are more aligned with the interests of the grid community as a whole than the other agents participating in the grid economy (see Table I).

Although the marketmaker agent can range from relatively simple to very complex, as long as it is acting in its selfish interests, its influence on the behavior of the grid is the same, with differences only in the degree of success. For the purpose of this paper, we have adapted a relatively simple, but complete implementation. The marketmaker agents are pursuing an investment strategy with the objective of achieving an investment target of having a uniform amount of resources for a fixed interval in the future. All the negotiation processes were single-step, without counteroffers. We have implemented a best-fit algorithm for the choices of resources to satisfy the requests.

In the following, we provide a more detailed discussion of the resource allocation model and communication mechanisms deployed in our system.

B. Resource allocation model

The components of the resource allocation model are the units of resource allocation, the allocation states and the actions through which the allocation states can be changed. The provider P has a set of available resources $R = \{r_1, r_2 \dots r_n\}$ where r_i is the available quantity of the resource class i. The main resource allocation unit in our model is the *usage chunk*, which is a triplet of $C =< t_{start}, t_{end}, R_c >$ where $R_c = \{r_{1c}, r_{2c}, \dots r_{nc}\}$ is a set of resource allocations. Usage chunks need to conform to a series of restrictions $\forall r_{ci} \in A_i$ where $A_i = \{a_{i1}, a_{i2}, \dots a_{ip}\}$ is the set of allocatable units of resource i. This covers the fact that hardware constraints limit the allocation of resources: for instance, on most Unix systems, memory can be allocated in chunks of 4KB. In addition to hard constraints, policy requirements may set the minimum limits on the size of allocatable resource chunks. At any given time, the sum of allocated resources can not exceed the available ones.

TABLE I
THE INTERESTS OF THE AGENTS PARTICIPATING IN THE RESOURCE ALLOCATION

Type	Complexity	Way to profit	Interest
Client	Low	-	Acquire resources at an acceptible price
Provider	Low	Being paid for resources	Maximize the income from allocated resources
Matchmaker	Low	Being paid for the matches	Increase the number of successful matches
Broker	Medium	Being paid for successful transactions	Increase the number of successful resource allocations
Marketmaker	High	Reselling resources for higher prices	Perform a better allocation of the resources.

The usage chunks can be in one of the five states:

`Free` the resources are not used and available for allocation

`Reserved` the resources are reserved for future allocation for the time of the negotiation. This state is strictly tied to a conversation and the user. The role of this state is to prevent multiple allocations of the same resources.

`Committed` the resource is allocated to the user for future use, but they are not in current use.

`In_Use` the resources are currently in use by a task

`Expired` the usage chunk occurs in the past ($t_{end} \leq t$). These allocation units might be kept around for accounting purposes, but they will be eventually garbage collected.

The provider maintains a database of available usage chunks. New usage chunks are created during the resource allocation process, on behalf of an external agent. The state machine describing the evolution of the states is described in Figure 1.

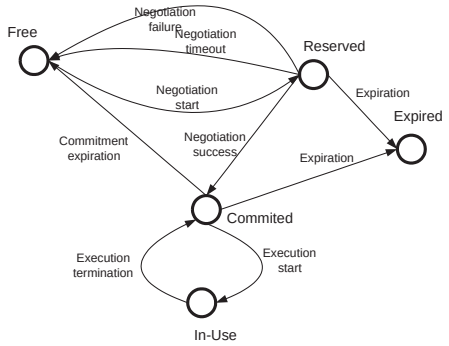

Fig. 1. The state machine describing the transitions between the various states of a usage chunk.

C. Communication mechanisms

Now, we turn towards the communication mechanisms which describe the operation of the virtual economy. In our descriptions, we will use the concept of a *subprotocol* [3] originally introduced in the Bond agent system [4]. A subprotocol is a set of *messages* exchanged between a set of *participants*, acting in well-defined *roles* to achieve a *goal*. The subprotocol is *closed* (messages in a subprotocol are answered with messages in the same subprotocol) and *undividable* (if a participants implements a role of the subprotocol, it needs to implement the role completely). A set of messages in a given subprotocol, exchanged between agents acting in the same roles is called a *conversation*. The same agent can participate in multiple conversations simultaneously, and it can have different roles in different conversations. For instance, a marketmaker can act as a buyer in some conversations and a seller in others.

1) Resource allocation subprotocol: The resource allocation subprotocol takes place between the two agents, in the roles of `Seller` and `Buyer`. The buyer has a set of resource requirements which he wants to satisfy. The resource requirements are specified as a collection of ranges which contain acceptable values. If the seller can satisfy the resource requirements, it will send one or more offers to the buyer. The offer contains the exact specification of the resource and its price. The buyer can accept the offer, cancel the negotiation process or send a counteroffer. After an accepted offer, a sale happens, the currency is transferred from the buyer to the seller and the customer receives a reservation id which it can use to access the resources.

2) Task execution subprotocol: The task execution subprotocol takes place between two agents in the roles of `Client` and `Executor`. The client submits a task for execution together with a reservation id for a set of resources. The reservation id was obtained as a result of the resource reservation subprotocol, although not necessarily between the same two agents. The `Executor` agent either executes the application itself or delegates the execution to further agents. The successful start of the execution is confirmed to the

Subprotocol id:	`Allocation`
Roles	`Buyer (B)` `Seller (S)`

Messages	`PriceQuery (B → S)`
	`PriceOffer (B ← S)`
	`NoOffer (B ← S)`
	`SaleAccept (B → S)`
	`CounterOffer (B → S)`
	`TerminateNegotiation (B → S)`
	`SaleConfirm (B ← S)`

Contracts	`SaleAccept` MUST be followed by `SaleConfirm` offers MUST be monotonic

TABLE II
A SUMMARY OF THE RESOURCE ALLOCATION SUBPROTOCOL

Subprotocol id:	`Execution`
Roles	`Client (C)` `Executor (E)`

Messages	`ExecutionRequest (C → E)`
	`ExecutionAccept (C ← E)`
	`ExecutionReject (C ← E)`
	`ExecutionFinished (C ← E)`
	`ExecutionQuery (C → E)`
	`ExecutionStatus (C ← E)`

Contracts	Execution MUST be started immediately if resource id is valid Execution finished MUST be reported immediately

TABLE III
A SUMMARY OF THE TASK EXECUTION SUBPROTOCOL

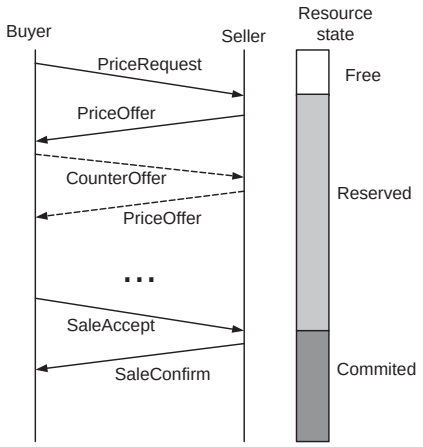

Fig. 2. A typical message sequence of the resource allocation subprotocol

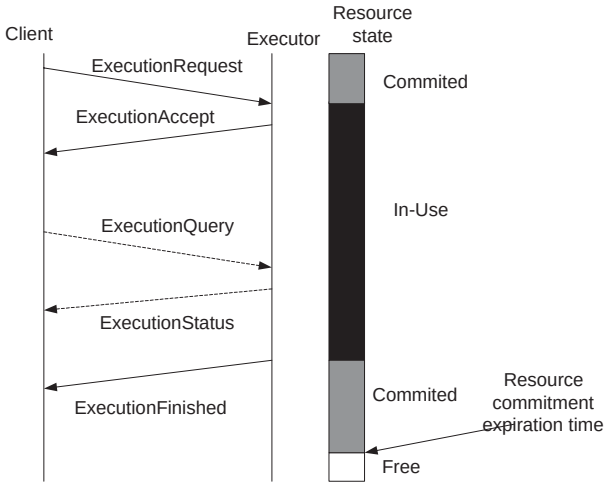

Fig. 3. A typical message sequence in the task execution subprotocol

`Client`. The only acceptable reason for not starting the execution is if the resource id does not identify a valid reservation of resources.

During the execution process, `Client` can periodically ask for the status of execution. The successful or unsuccessful termination of the execution is reported by messages taken on the `Executor`'s initiative.

III. EXPERIMENTAL RESULTS

A. Experimental framework and scenarios

In the following, we describe a series of experiments comparing the behavior of a market grid environment in three different scenarios involving matchmaker, broker, and market-maker agents. We have simulated the grid market environment using the agent module of the YAES simulation framework [5]. Our goal was to simulate the functioning of the agent grid in as much detail as possible, including the cost associated with the agent communication. The agent communication was implemented using a FIPA ACL conforming message format, while the message delivery was performed with a local directory with a simulated latency. We had chosen not to implement the details regarding security, authentication and accounting. While these are important components of a grid environment, they add a constant overhead to communication costs, and thus does not affect our results significantly.

The experiments were grouped in three main scenarios discussed below.

1) Scenario CMP: Customers + Matchmaker + Providers: In the first series of experiments we had a configuration of customers, providers, and a matchmaker. A variable number of clients are having an infinite supply of tasks of various length and resource requirements, which they execute sequentially. First, the clients contact the matchmaker agent using the `Query` subprotocol for a collection of providers which are a good match for the task. The clients then start negotiations with all the returned matches using the `Allocation` subprotocol. From all the returned offers, the client selects the one which provides the highest satisfaction (considering its utility and price), and pays for the allocation of resources. The other negotiations are terminated. After allocation, the client immediately sends the task for execution using the `Execution` subprotocol. If no offers were returned, the client immediately restarts the process of searching for providers from the matchmaker. After the termination of a task, the client

immediately moves to the next task.

We should note that in the CMP scenario, the clients have extensive responsibilities, including directly contacting and negotiating with the providers. The general setup of the scenario is described in figure 4.

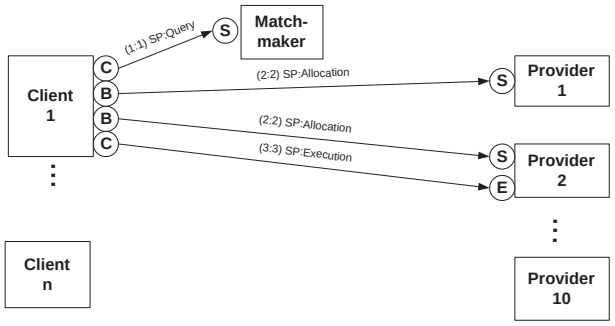

Fig. 4. The experimental setup for the Customers + Matchmaker + Providers scenario. The lines represent conversations performed according to a given subprotocol, with the arrow indicating the direction of the first message. The circles at the end of the lines mark the role played by the agent in the given conversation. The (i:j) numbers on the lines show that the conversation was the i-th conversation started and the j-th conversation finished.

2) Scenario CBMP: Customers + Broker + Matchmaker + Providers: In the second series of experiments, we had a configuration of customers, providers, a broker, and a matchmaker. In this scenario, the clients contact a broker agent, using the `Allocation` subprotocol. The broker contacts the matchmaker agent for matches, and starts a set of negotiations with the returned providers using the same `Allocation` subprotocol. The resulting offers are first pre-processed by the broker, by discarding the dominated offers (with the same or lower resource offering and the same or higher price) and randomly selecting one from the equivalent offers. The resulting set of offers are forwarded to the client, while the negotiations with the discarded providers are terminated without further contacting the client. The client selects the best offer according to its satisfaction function. The resources are allocated at the request of the broker, the provider and the broker are paid, and the reservation id is forwarded to the client. At this point, the role of the broker ends. The clients are directly contacting the providers using the `Execution` subprotocol for the execution of the tasks. The general setup of the experiment is described in Figure 5.

3) Scenario CKMP: Customers + MarKetmaker + Matchmaker + Providers: In the third series of experiments we had a configuration of customers, providers, a marketmaker and a matchmaker. In this scenario, the clients are contacting the marketmaker agent for the execution of tasks. The marketmaker has a separate ongoing thread of execution, in which it autonomously searches for investment opportunities, negotiating and allocating resources on the providers. These negotiations are conducted using the `Allocation` subprotocol and their results are stored in the allocated resource database in the marketmaker. The marketmaker prefers to acquire large resource chunks (in terms of time and available resource). One such allocation normally corresponds to several

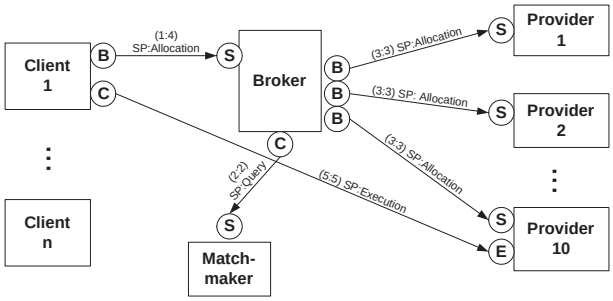

Fig. 5. The experimental setup for the Customers + Broker + Matchmaker Providers scenario. The lines represent conversations performed according to a given subprotocol, with the arrow indicating the direction of the first message. The circles at the end of the lines mark the role played by the agent in the given conversation. The (i:j) numbers on the lines show that the conversation was the i-th conversation started and the j-th conversation finished.

tasks being executed.

When the client consults the marketmaker using the `Allocation` subprotocol, the marketmaker checks if it can satisfy the request, and generates an offer for the resource allocation. While the broker only passes a filtered set of offers to the client, the marketmaker's offer is decoupled from the price at which the marketmaker acquired the resource. Moreover, the marketmaker at any given time has several requests outstanding and it applies an allocation policy for fitting them with the allocated resource database. In our experiments, we set this policy such that the marketmaker attempts to fit the larger tasks first and uses a best-fit choice for each individual task.

Once the offer is accepted by the client, it receives a special reservation id, which is locally generated by the marketmaker. Then the client sends the task for execution to the marketmaker using the `Execution` subprotocol and this reservation id. The marketmaker forwards the tasks to a provider using a subset of one of its own previous allocations. We note that in this scenario the client is never directly communicating with the providers. The general setup of the experiment is described in Figure 6.

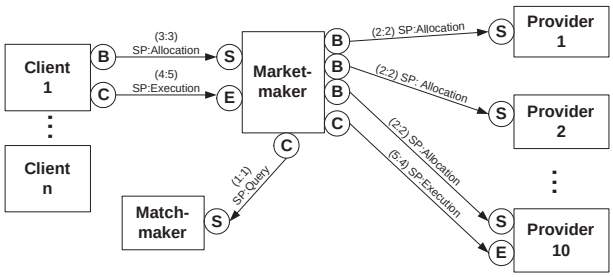

Fig. 6. The experimental setup for the Customers + Marketmaker + Matchmaker + Providers scenario. The lines represent conversations performed according to a given subprotocol, with the arrow indicating the direction of the first message. The circles at the end of the lines mark the role played by the agent in the given conversation. The (i:j) numbers on the lines show that the conversation was the i-th conversation started and the j-th conversation finished.

B. Simulation parameters

We run an extensive series of simulations with the above scenarios. The number of providers were fixed at 10, while the number of customers was varied from 2 to 60. We used two types of tasks: small tasks with an average resource requirement of 80 resource equivalent units (REU's) and large tasks with an average resource requirement of 180 REUs. The number of large tasks was 30% of the total requests. The resource requirements of the tasks were normally distributed around their mean. The task execution time was also normally distributed with a mean of 100 seconds. There were two types of providers, 7 providers with the total capacity of 100 and 3 providers with 200 REUs each. A 1 second messaging latency was assumed for all communications.

All agents were provided with a sufficient amount of money for the execution of their tasks, thus the lack of money was not a factor in limiting the performance of the system. This choice of course did not affect the strategies of the agents, which were set such that to maximize the profit (for providers and marketmakers) or maximize satisfaction (for the clients). For each scenario (CMP, CBMP and CKMP) we run 30 simulation rounds spanning a simulation timeframe of 5000 seconds each. For every measurement, we computed the mean values, variance, and 95% confidence intervals.

C. Results

1) Tasks executed: We have separately counted the total number of tasks and the total number of large tasks executed during the simulation. The total number of tasks (Figure 7a) shows a similar evolution for the three scenarios. For a partially loaded system, the results are roughly identical, with the results of the broker approach being somewhat lower, mainly due to the overhead associated with the mediated strategy. At high loads however, the marketmaker approach delivers an approximately 20% lower number of total tasks.

The reason for this behavior becomes clear if we check the graph for the large tasks in Figure 7b. By employing a best-fit approach over a group of requests, the marketmaker approach manages to maintain the number of executed large tasks at approximately the proportion of these tasks in the flow of requests. For the CMP and CBMP approaches, where each request is handled individually, such a fitting process is not possible, and the number of executed large tasks is declining with increasing load. The reason for this is that the larger number of small tasks are starving the larger tasks by partially occupying resources. This is a clearly undesireable behavior, which cannot be solved in the CMP and CBMP scenarios, given the interests of the participants.

In the CMP scenario, the interest of the matchmaker is to return as many scenarios as possible, while the interest of the clients is to have their tasks executed. One can envision matchmaking strategies which would not return matches which are "too large" for the request, such that these would be reserved for possible large tasks. But this behavior is *not* in the interest of the matchmaker or the provider, and thus we cannot expect it to happen in a real market environment. Similarly, for the CBMP case, the broker is interested in optimizing the current transaction for its client, because its economic well-being depends on the success of the transaction.

We need to note that the problem is not with the relatively simple algorithms employed by the middle agents in the simulation. More sophisticated algorithms would simply make the agents more successful in pursuing their interests. Thus, if the problem can be traced to the agents' interest not being aligned with the interest of the grid as a whole, more sophisticated algorithms would exacerbate the problem.

In the case of the marketmaker agent, the interest of the agent is to efficiently allocate the resources it previously purchased. The "best-fit/largest-first" algorithm we employed is a relatively crude solution, but it is in-line with the interests of the grid as a whole.

2) Resource utilization: Figure 8 shows the average used resources on the providers for the three scenarios. At high loads, this utilization is limited to about 80%. This value is limited by the mix of tasks and the problem of fitting them to the resources of the providers. The CKMP model provides the best performance because it is the only one providing a fitting algorithm. The CBMP scenario has the lowest performance due to the broker's messaging overhead.

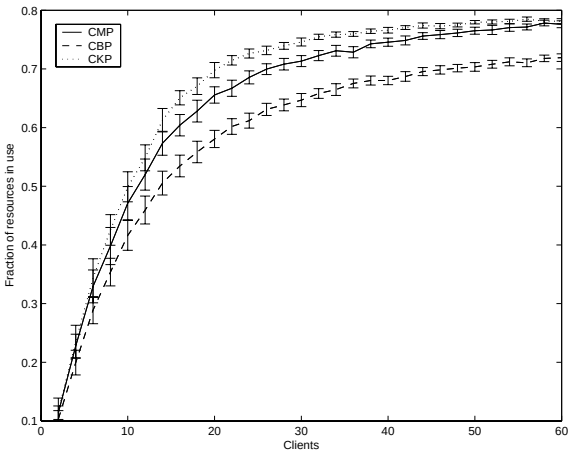

Fig. 8. The resources currently in use by tasks in function of the number of clients. The bars indicate the 95% confidence intervals.

3) Number of messages passed and scalability: The total number of messages passed on the simulation is shown in Figure 9. Essentially, for heavily loaded networks, the marketmaker scenario has an order of magnitude smaller number of messages than the broker and the matchmaker model. This is the consequence of two facts: (1) the marketmaker buys resources in larger chunks, thus eliminating a large number of messages and (2) the offer generation for a marketmaker happens internally, without the need to contact the providers.

We note that the increase in the number of messages for the CMP and CBMP models is due to retries after unsuccessful allocations. Of course, it is easy to limit the number of messages per time unit, for instance by imposing a mandatory wait behavior. This behavior however is not in the interest of the client, matchmaker, or broker and it needs to be imposed externally. The best interest of these agents is an immediate

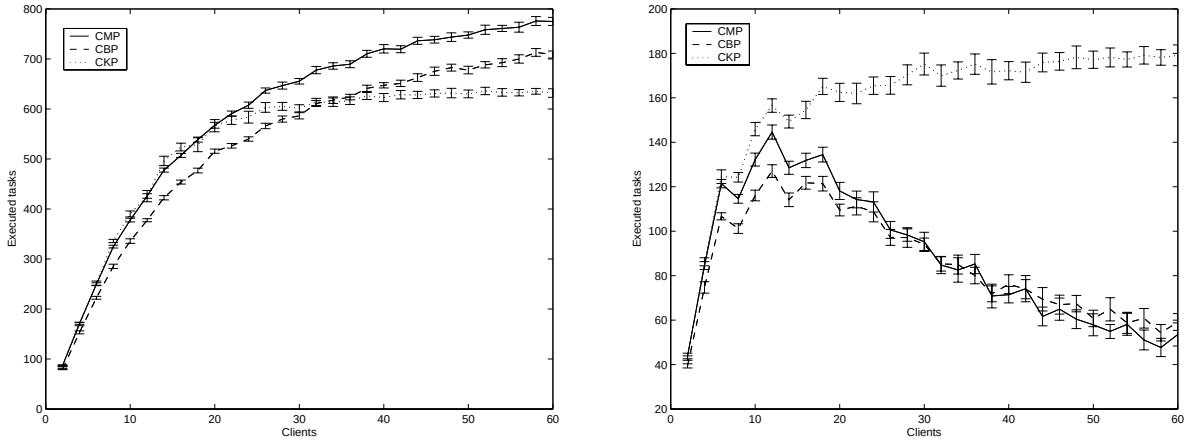

Fig. 7. (a) The total number of tasks executed and (b) the number of large tasks executed. The bars indicate the 95% confidence intervals.

retry. For the marketmaker however, there is no incentive in a retry for any of the parties.

As opposed to the superficially resembling problem of Carrier Sense Multiple Access Collision Detect (CSMA-CD) systems, frequent retries do not lead to collisions and do not diminish the throughput of the system. Therefore there is no incentive to delay the retry, neither at the individual, nor at the group level. Such incentives need to be built into the "rules of encounter" of the market, for instance by making the clients pay even for unsuccessful negotiations. The study of such systems, however, are outside the scope of this paper.

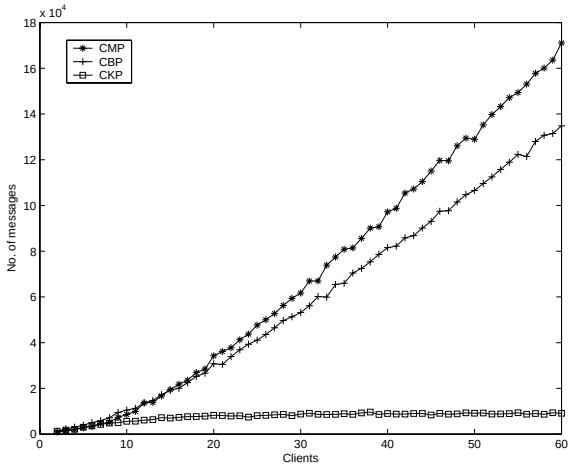

Fig. 9. The total number of messages passed during the simulation, in function of the number of clients. The bars indicate the 95% confidence intervals.

IV. Previous Work

Economic models are frequently considered paradigm in the resource management of distributed systems. The types of resources considered by economic models cover: CPU cycles, main memory, storage, network activity, signals received, software, and libraries accessed [7]. The economic models appear in a number of variants.

In the *commodity market* model, resource providers advertise their resource prices and charge users based on the amount of resource used. The pricing policy could be based on a flat fee, the resource usage duration, the subscription, and the demand-and-supply [11]. In the *posted price* variant, the model allows providers to advertise special offers to attract users. An approach based on *bargaining*, that is direct negotiation between costumers and providers is used when there is no clear demand-and-supply relationship and globally established price. A special case of the bargaining model is the *tendering/contract-net* approach. The auction models allow customers to bid for resources according to one of the well known auction models: 1) English auction; 2) first-price sealed-bid auction; 3) Vickrey auction (second-price sealed-bid); and 4) Dutch auction. Examples of auction based systems include Spawn [19] and Popcorn [12]. In *Bid-based Proportional Resource Sharing* the percentage of resources allocated to the user is proportional to his bid in comparison to other users' bids. In the *Community/Coalition/Bartering* model, a community of resource owners share each other's resources. Those resource owners contribute to the community get credits by sharing their resources. The credit of a resource owner decides how much resources he can get from others. Case systems include Condor [23], SETI@home [24], and Mojo Nation [21].

The nature of economic models which imply self-interested participants with some level of autonomy in the decisions, makes an agent based approach a natural fit. In general, approaches based on direct negotiation between customers and providers are suitable only for small numbers of agents. As the size of the agent society grows, it becomes necessary to introduce *middle agents*, which facilitate the interaction between large sets of providers and customers. The most important type of middle agents discussed in the literature of economic models in distributed systems are the *matchmaker* and *broker* agents [9], [20].

The problem of matchmaking has been extensively studied in the field of multi-agent systems. The notable results in this area are ACLs (Agent Communication Languages) and matchmaking algorithms based on these languages, such as ABSI [16], COIN [10], InfoSleuth [2], LARKS (Language for Advertisement and Request for Knowledge Sharing) [18], ITL (Information Terminological Language) [17]. Research for service discovery in the Internet involves ontology-based matchmaking. In [13], a semantic matchmaking framework based on DAML-S was proposed for semantic matchmaking of web services capabilities. The matchmaking framework of Condor [15] system uses a semi-structured data model called classified advertisements ("classads") to describe the resources and requests for matchmaking. An ontology-based grid resource matchmaking framework is introduced in [1].

In contrast, relatively little work was done on the broker agents. A resource broker in Nimrod/G uses a scheduling mechanism driven by a user specified application deadline and a resource access budget; an infrastructure called GRACE (GRid Architecture for Computational Economy) was proposed to provide dynamic resource trading services [6]. Three adaptive scheduling algorithms for Nimrod/G application level resource broker are discussed in [8]: a time minimization algorithm that attempts to complete the execution as early as possible with the budget constraint, a cost minimization algorithm that attempts to complete the execution as economically as possible before the deadline, and a none minimization algorithm that attempts to complete the execution before the deadline with the budget constraint without minimizing execution time and money paid.

V. Conclusions

In this paper, we considered a market based grid architecture and introduced a new type of middle agent, the marketmaker. We presented its architecture, goals and strategies and contrasted it with two well known middle agent types, the matchmaker and broker agents. In a series of simulations, we find that the use of marketmaker agent leads to a better performance in the allocation of large tasks and a significant reduction of the messaging overhead. The main conclusion of our research is that while the quality of the deployed algorithms of the agents can make important quantitative differences, the general behavior of the system is determined by the selfish interest of the coordinating agents. The marketmaker agent profits by improving the efficiency of resource allocation, and thus can represent an important component in future grid systems.

References

[1] X. Bai, H. Yu, Y. Ji, and D. C. Marinescu. Resource matching and a matchmaking service for an intelligent grid. *International Journal of Computational Intelligence*, 1(3):197–205, 2004.

[2] R. J. Bayardo, Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 26,2, pages 195–206, New York, 13–15 1997. ACM Press.

[3] L. Bölöni, R. Hao, K. Jun, and D. C. Marinescu. An object-oriented approach for semantic understanding of messages in a distributed object system. In *Proceedings of the International Conference on Software Engineering Applied to Networking and Parallel/ Distributed Computing, Rheims, France*, May 2000.

[4] L. Bölöni and D. C. Marinescu. An object-oriented framework for building collaborative network agents. In H. Teodorescu, D. Mlynek, A. Kandel, and H.-J. Zimmerman, editors, *Intelligent Systems and Interfaces*, International Series in Intelligent Technologies, chapter 3, pages 31–64. Kluwer Publising House, 2000.

[5] L. Bölöni and D. Turgut. YAES - a modular simulator for mobile networks. In *Proceedings of the 8-th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems MSWIM 2005*, pages 169–173, October 2005.

[6] R. Buyya, D. Abramson, and J. Giddy. Economy driven resource management architecture for computational power grids. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2000)*, 2000.

[7] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service-oriented grid computing. In *Proceedings of the 10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, page 83, April 2001.

[8] 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 *Proceedings of the 2 nd International Workshop on Active Middleware Services (AMS 2000)*. Kluwer Academic Press, August 2000.

[9] M. Klusch and K. P. Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 197–224. Springer, 2001.

[10] D. Kuokka and L. Harada. Matchmaking for information agents. In *IJCAI (1)*, pages 672–678, 1995.

[11] L. W. McKnight and J. Boroumand. Pricing internet services: Approaches and challenges. *IEEE Computer*, 33(2):128–129, 2000.

[12] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet - the POPCORN project. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 592, Washington, DC, USA, 1998. IEEE Computer Society.

[13] M. Paolucci, N. Srinivasan, K. P. Sycara, and T. Nishimura. Towards a semantic choreography of web services: From WSDL to DAML-S. In *Proceedings of the First International Conference on Web Services (ICWS'03)*, pages 22–26, 2003.

[14] T. Payne, R. Singh, and K. Sycara. Facilitating message exchange through middle agents. In *The First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002.

[15] R. Raman, M. Livny, and M. H. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 140–146, 1998.

[16] N. Singh. A common Lisp API and facilitator for ABSI: version 2.0.3. Technical Report Logic-93-4, Logic Group, Computer Science Department, Stanford University, 1993.

[17] K. Sycara, J. Lu, and M. Klusch. Interoperability among heterogeneous software agents on the internet. Technical Report CMU-RI-TR-98-22, Carnegie Mellon University, PA (USA), 1998.

[18] K. P. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.

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

[20] H. C. Wong and K. P. Sycara. A taxonomy of middle-agents for the internet. In *ICMAS*, pages 465–466, 2000.

[21] Mojo Nation. URL http://www.mojonation.net/.

[22] F. Zeuthen. *Problems of Monopoly and Economic Warfare*. Routledge and Sons, 1930.

[23] CONDOR. URL http://www.cs.wisc.edu/condor/.

[24] SETI@home. URL http://setiathome.ssl.berkeley.edu/.