Agents for Market-Based Computational-Resource Allocation

Brecht Reggers[a] Floris Wiesman[b] Nico Roos[a]

[a] IKAT, Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht
[b] Dept. of Medical Informatics, Academic Medical Center, P.O. Box 22700, 1100 DE Amsterdam

Abstract

Distributed computing is a well-known approach to perform computationally intensive tasks. A problem inherent to this approach is the allocation of tasks to machines. This problem can be successfully addressed with a market mechanism. The article presents a comparison between two market mechanisms for an agent-based trading platform for computational resources. The platform consists of the following agents: (A) management agents (the directory facilitator, the controller agent, the account agent, and the bank agent), (B) client agents, and (C) one mediator agent. The market mechanisms used were an unmediated one, Contract Net, and a mediated one, the sealed bid, periodic, uniform-price, double auction. Experiments showed that overhead and gain were quite similar for the two market mechanisms. Regarding scalability – a highly important factor for large distributed projects – the auction scored clearly better than Contract Net.

1 Allocation of computational resources

Over the years, the computational problems around the globe have increased in size and complexity. Current examples of complex problems are the search for a new smallpox cure and the human genome project. Although the computing power per machine grows rapidly too, it is still hard to solve the problems mentioned in an acceptable time frame. A currently popular approach that attempts to increase the available computing power is to use a set of computers, connected by a network, enabling them to work on the same problem. This approach is called *distributed computing*. Because networks, in particular the Internet, are heterogeneous environments with a diversity of machine types and operating systems, there is a definitive need for an approach that can find machines offering computing resources, assign tasks to these machines, and collect the results. Moreover the approach must cope with an unknown and ever-changing environment. In this article we describe an approach using a multi-agent system to perform the tasks and a market mechanism to distribute tasks over agents. We investigate which market mechanism is most suitable to be used with respect to generated overhead, scalability, and performance gain.

Section 2 provides a brief overview of related work, Section 3 describes our approach, Section 4 shows the results of the experiments, and Section 5 summarizes our conclusions.

2 Related work

Much research has been carried out concerning distributed computing, as well as concerning agents. However, little research has combined these two subjects. Below we describe two such projects: Messor and YACA.

The Messor algorithm in combination with the ANTHILL system is a system for distributed computing, based on the metaphor of colonies of ants [4]. Each ant carries a computation task to be performed, and only drops it of at a computer for execution after it has wandered randomly for some time without encountering other tasks. Colonies of such ants try to disperse tasks uniformly over their environment; as such, they can form the basis for a load-balancing algorithm. This approach is highly scalable, yet it may be inefficient with respect to the generated overhead.

YACA [3] is a distributed computing platform that uses four managing agents and a number of mobile client agents. The management agents manage the machines connected to the network: (1) The Directory Agent keeps track of the machines belonging to the cluster and provides a white-pages service to which participating agents must register. (2) The Controller Agent manages access to a single machine and controls the migration of client agents in case the machine becomes overloaded. To do its job, the Controller Agent is assisted by (3) the Weather Agent, who monitors the resources of the machine, and (4) the Account Agent, who monitors the resources used by client agents. YACA currently lacks good ways to organise the recompensation (e.g., financially) of resources and efficient algorithms to allocate machines with spare resources to computing agents.

3 Markets for computational resources

In this section we first describe the two market mechanisms used: the SPUD auction and Contract Net. We continue with the design goals, the design of the agents, and a discussion on computing resources.

3.1 SPUD auction

In a study carried out by Braun [2] it is argued that the most suitable market mechanism for extending the YACA platform (see Section 2) is a sealed bid, periodic, uniform-price, double auction (which we will call a SPUD auction). In other words: the bids are not visible to other participants, the auction is to be held at intervals, and has to support multiple buyers and sellers at the same time. Also, when multiple units are being auctioned, the price for all units sold is equal for all winners.

The SPUD auction works as follows. After the auctioneer has initiated an auction by starting a bidding round, the sellers (M) and the buyers (N) can all submit one sealed bid. This way, $L = M + N$ bids are collected. A specified time interval later, the auctioneer ends the auction, and the price at which the goods are sold – the clearing price – is calculated. A clever way to obtain a clearing price in an auction which does not involve further negotiation is amplified in [6, 8]. The technique used is called the $(M + 1)$st-price rule.

The $(M + 1)$st-price auction clearing rule sets the price at the $(M + 1)$st highest value among all L bids. This implies that the $(M + 1)$st-price is undefined if there are no buyers.

Once the clearing price has been calculated, the auctioneer has to determine which bids are going to be matched: the *transaction set*. This is done as follows: let m denote the number of unit sell offers at or below the clearing price, and n the number of unit buy offers at or above the clearing price. Let $a = \min(m, n)$. The transaction set consists of the a highest unit buy offers and the a lowest unit sell offers. The sell offers and buy offers in the transaction set may be matched arbitrarily.

The $(M + 1)$st-price rule is incentive compatible for the buyers, so bidding one's private value is the dominant strategy.

3.2 Contract Net

The Contract Net (CNET) protocol is a protocol for achieving efficient cooperation through task sharing [7]. To distribute pending tasks, CNET relies on direct communication between the participants. For CNET, there are two different types of agents, an initiator and a participant. At any time, any agent can be an initiator, a participant, or both. CNET creates a means for contracting as well as subcontracting tasks; in this sense initiators are managers and participants are contractors. An initiator is an agent willing to buy some task; participants are agents wanting to sell the task.

The protocol is composed of a sequence of four steps. The agents must go through the following loop of steps to negotiate each contract. (1) The initiator sends out a Call for Proposals (CFP). The CFP includes units, price, and a deadline by which the participants must respond. (2) Each participant reviews CFPs and bids on the feasible ones accordingly. (3) After the deadline, the initiator chooses the best bid and awards the contract to the respective participant. The winning participant must now attempt to expedite the task, which may mean generating new subtasks. (4) The initiator rejects the other bids.

3.3 Design goals

For the distributed computing system to be developed, we set the following four design goals.

1. The system should use a mobile multi-agent platform (because of flexibility and suitability for market mechanisms).
2. Two market mechanisms operating on this platform should be constructed (viz., an unmediated and a mediated mechanism, which serve as two extremes that we would like to compare).
3. A low overhead (so as to realize an efficient system).
4. Operating-system independency (since a distributed computing system must be able to cope with the diversity of operating systems on the Internet).

To achieve these goals, a set of agents has been created. Their internal workings and the communication between these agents will be addressed in the next section. The agents have to run on a framework. JADE [1] was chosen for different reasons. The most important reason was its support for mobile agents. Because of the distributed nature of our project, agent mobility is a *sine qua non*. JADE also provides good means for

communications between agents; all communication inside JADE fully supports the FIPA standards. Furthermore, JADE has been built on top of Java so it is compatible with most operating systems, which satisfies our fourth design goal.

3.4 Design of the agents

Our entire system, including the market mechanisms, is composed of agents. Three categories of agents can be distinguished: (A) the management agents (the directory facilitator, the controller agent, the account agent, and the bank agent), (B) the client and sub-client agents, and (C) one mediator agent. The agents are explained below; for full details on the design and implementation we refer to [5].

3.4.1 Management agents

The **directory facilitator** (DF) can be seen as a yellow-pages service on which agents can search for other agents that offer services. Its main task is managing a list of all the agents that exist in the cluster. The DF is part of the FIPA specifications covering agent management for inter-operable agents.

The **controller agent** is the main manager of a node. On every single node a controller agent has to be present. The controller agent uses preconfigured settings (i.e., the maximum number of sub-clients to be admitted, the price-speed ratio and the speed of the machine) to calculate the price of the resources offered. The user can design a pricing policy for the controller agent.

Once a deal has been made with a sub-client agent in need of computing resources, the controller agent gives permission to that sub-client agent to come to the controller's node and to start its calculation. After the sub-client agent has completed its calculation the controller agent must see to it that the sub-client concerned actually leaves the node.

The purpose of the **account agent** is to keep track of the resources used by the visiting sub-client agents on a particular node. At the end of the sub-client agent's stay at the node, the account agent sends an invoice to the sub-client agent.

The **bank agent** is an independent service that acts like a simplified real-world bank. Agents use is to store and transfer credits.

3.4.2 Client agents

Client agents are the 'workers' of the system. They carry a program with them (which is the task they want to complete), buy computing resources from a node, travel to that node, and start computing. While some nodes may have no clients present at a given time, other nodes may have multiple clients. Their precise behaviours and internals on how to accomplish these tasks are heavily dependent on the market system used. Therefore, each type of market system requires a different set of behaviours for the client agent.

Computing tasks are given to the client agents directly by the user. When given a task, the client agent generates as many sub-clients as are needed to complete the task. Even if only one client is needed to carry out the task a sub-client agent is generated. Generating subclients is left to the programmer; it is not a part of our system. Now a modest hierarchy arises: a parent-client with a number of sub-clients. The sub-clients

all receive a part of the task. The parent-client itself does not participate in the search for computing resources, nor does it perform calculations. Instead it is an immobile agent that receives the results from the sub-clients and eventually presents the user with the final result.

3.4.3 Mediator agent

The mediator agent occupies a central role in the mediated market mechanism. It manages all sell and buy offers of the participants. The foundation of the mediator agent is the SPUD auction protocol in conjunction with the $(M + 1)$st price rule.

3.5 Computing resources

In the previous sections we have used the term *computational resource* without defining it. We now state that computational resources can be considered broadly as incorporating machine properties such as (i) processor time and speed, (ii) hard disk space and speed, (iii) amount and speed of RAM, and (iv) also network properties such as the speed of the connections. Since the vast majority of tasks carried out on a distributed computing system are computationally intensive and thus rely mainly on the speed of the processor, our system uses the assumption that the speed of a machine is the defining computational resource. We chose the FLOPS (Floating Point Operations Per Second) unit to benchmark the speed of the machines in our system.

We can now address the question how computational resources can be offered for sale. In our system this is done by offering resources for a period of time (e.g., a second). When selling resources, a 700 MFLOPS machine (with a load of 0%)[1] can offer 700 MFLOPS for sale, and the price will be per second. Since the computing resources offered represent speed, a buying sub-client agent can decide to spend more money on a faster machine if it has a task that is to be completed quickly. The amount of money a sub-client agent can spend is specified by the parent-client agent to which it belongs. The parent-client agent in turn receives this figure from the user when the task is assigned, and later from the bank agent.

4 Experiments

In this section we present the general setup of the experiments (Subsection 4.1), and the setup and results of the series of experiments concerning overhead (Subsection 4.2), scalability (Subsection 4.3), and gain (Subsection 4.4). Full details are provided in [5].

4.1 Experimental setup

For the experiments a cluster of five computers connected by a 100 Mbit Ethernet LAN was created. The speeds of the computers ranged from 444 to 3244 MFLOPS.

For the test program we chose a computationally intensive task, which could be divided in a straightforward manner. The program counts the number of primes in a given

[1] This is a theoretical value in which system processes have not been taken into account.

Table 1: Consumption of the SPUD auction mechanism relative to the CNET mechanism.

Interval	1/single	3/single	5/single	1/network	3/network	5/network
0–20,000,000	99.59%	103.09%	103.94%	99.22%	104.64%	104.70%
0–40,000,000	99.86%	102.53%	102.58%	98.89%	104.02%	103.09%
0–60,000,000	99.88%	100.85%	100.64%	96.82%	103.52%	101.34%
0–80,000,000	99.61%	101.65%	100.77%	97.87%	103.99%	100.40%
0–100,000,000	99.87%	100.63%	100.53%	101.33%	101.73%	101.28%

interval without improving the efficiency of the program using the prime numbers found. When the task is divided over multiple sub-clients, all sub-clients receive intervals of equal sizes. Every experiment was run ten times; the results were averaged.

4.2 Overhead

We define the overhead as the consumption of the system software (including the operating system and any utility supporting the actual application); we define consumption as *seconds used · speed of host*.

For each of the two market mechanisms the overhead was established for various setups. Different numbers of sub-clients were used and a difference was made between running on a single node and running on the network. Table 1 shows the overhead in consumption of the SPUD auction mechanism as a percentage of the overhead of the CNET mechanism. Hence, values greater than 100% indicate SPUD has consumed more resources than CNET. Despite their different natures, the mechanisms show small differences in overhead.

4.3 Scalability

Scalability expresses the ability to expand the number of users or increase the capabilities of a computing solution without making substantial changes to the systems or application software. In our system, the two factors that need to be increased so as to simulate a real-world application are (1) the number of nodes that serve as a host for the sub-client agents, and (2) the number of sub-client agents. Because of the wide range of possibilities we restricted the experiments to the following agents/hosts setups on the 1–60,000,000 interval only: 5/5 (5 sub-client agents with 5 hosts), 10/10, 15/15, 20/20, 25/25, 30/30, and 35/35.[2]

Figure 1 depicts the time used for negotiation for the different setups. The vertical lines on the points of measurement reflect the variation coefficients. It is clear that as the number of agents and hosts grows, the negotiation time for CNET increases faster than for SPUD. This effect is even stronger for the total number of messages sent and received by all sub-clients that are working on the same task (not shown here). The effect can be explained by the number of communication links required during negotiation: with CNET this is $s \cdot h$ (where s is the number of sub-clients and h the number of hosts), whereas with SPUD, it is only $s + h$, thanks to the mediator agent.

[2] As stated in Subsection 4.1, our network counts five computers. For the scalability experiments, multiple hosts were allowed to reside on one machine.

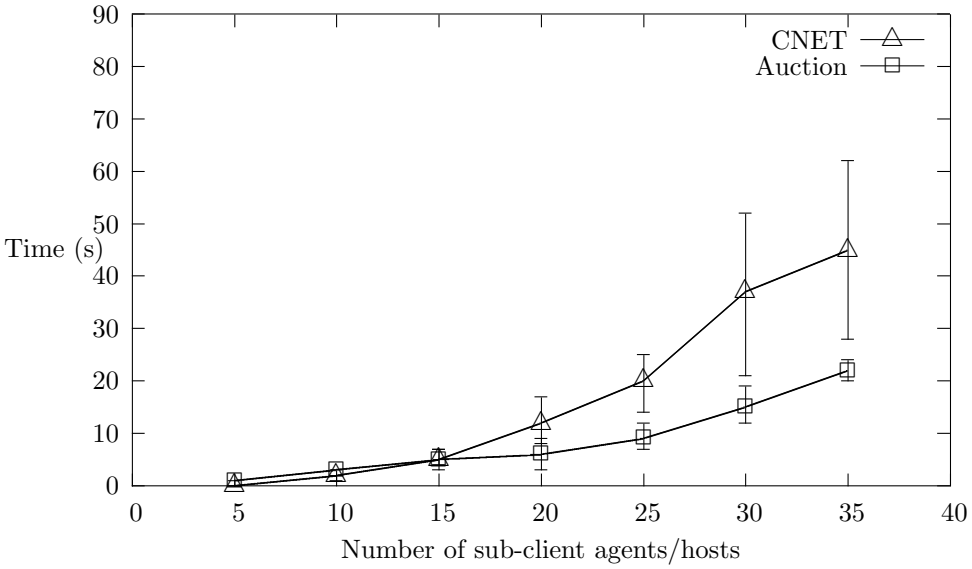

Figure 1: Time used for negotiation with CNET and SPUD auction mechanisms.

Table 2: Dual-node gain of the SPUD auction mechanism and the CNET mechanism.

Interval	Auction	CNET	Difference
0–20,000,000	48.09%	49.20%	1.12%
0–40,000,000	49.84%	49.11%	-0.72%
0–60,000,000	48.62%	49.83%	1.21%
0–80,000,000	49.91%	49.63%	-0.28%
0–100,000,000	49.37%	49.96%	0.59%

4.4 Gain

In our experiments we use the gain to measure how much faster our distributed system is compared to a non-distributed (single-node) system. Since the machines in our experiments are not identical, we computed the gain using consumption instead of time used.

To examine the single-node system, the same task was executed twice in a row. To examine the dual-node system, the same task was executed twice, but this time the tasks were 'released' together. Each host was set to allow only one sub-client agent at a time. This procedure was executed five times for all intervals. Afterwards, the mean of the single-node case was computed, as well as that of the dual-node case, but for the latter we computed one mean for each machine. These dual-node means represent the consumption of one half of the task. Of the two means from the dual-node system, the highest one was selected and compared to the mean of the single-node system.

Table 2 shows the dual-node gains for the two market mechanisms. All the gain percentages approximate the theoretical upper bound of 50%. No trend is visible: the results fluctuate slightly but not significantly.

5 Conclusions

In this article we have presented a multi-agent system for market-based allocation of computational resources. We have investigated two different market mechanisms: a mediated mechanism (the SPUD auction) and a unmediated mechanism (the Contract Net).

Despite their dissimilar nature, both mechanisms proved to perform well. The results of the experiments concerning overhead and gain actually match quite consistently. However, in one area of high importance for large distributed projects, namely scalability, the CNET mechanism clearly had to acknowledge the superiority of the auction mechanism. Therefore, we may conclude that in our comparison, the SPUD auction mechanism is most suited to be used on an agent-based trading platform for computing resources for distributed computing.

In our experiments, the sub-client agents have sufficient money to their disposal to be able to use the fastest host available. In future research we aim to investigate how effective market mechanisms are when sub-client agents are less prosperous. Also, the determination of the selling prices on the hosts is currently realised straightforwardly, so more efficient pricing mechanisms can be conceived. More complex price-related factors on the side of both buyers and sellers can be the foundation for (adaptive) strategies for both parties.

References

[1] F. Bellifemine, A. Poggi, and G. Rimassa. JADE: A FIPA-compliant agent framework. In *Proceedings of PAAM '99*, pages 97–108, London, England, April 1999.

[2] L.M.M. Braun. Extending the Yaca platform by market mechanisms. Technical report, IKAT, Universiteit Maastricht, Maastricht, The Netherlands, December 2001.

[3] S. Kleijkers, F. Wiesman, and N. Roos. A mobile multi-agent system for distributed computing. In *International Workshop on Agents and Peer-to-Peer Computing*, volume 2530 of *LNCS*, pages 158–163, 2003.

[4] A. Montresor, H. Meling, and O. Babaoglu. Messor: Load-balancing through a swarm of autonomous agents. In *International Workshop on Agents and Peer-to-Peer Computing*, volume 2530 of *LNCS*, pages 158–163, 2003.

[5] B. Reggers. Markets for computational resources. Master's thesis, IKAT, Universiteit Maastricht, Maastricht, The Netherlands, 2004. CS–04–02.

[6] M. Satterthwaite and S. Williams. Bilateral trade with the sealed bid k-double auction: Existence and efficiency. *Journal of Economic Theory*, 48:107–133, 1989.

[7] R.G. Smith. The contract net: a formalism for the control of distributed problem solving. In *Proceedings of the 5th International Joint Converence on Artificial Intelligence*, Cambridge, Massachusetts, 1977.

[8] P.R. Wurman, W.E. Walsh, and M.P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24(1):17–27, 1998.