Engineering Large-scale Distributed Auctions *

(Short Paper)

Peter Gradwell
University of Bath
Dept. of Computer Science
Claverton Down, Bath, UK
pjg@cs.bath.ac.uk

Michel Oey
Vrije Universiteit Amsterdam,
Dept. of Computer Science,
The Netherlands
michel@cs.vu.nl

Reinier Timmer
Vrije Universiteit Amsterdam,
Dept. of Computer Science,
The Netherlands
rjtimmer@cs.vu.nl

Frances Brazier
Vrije Universiteit Amsterdam,
Dept. of Computer Science,
The Netherlands
frances@cs.vu.nl

Julian Padget
University of Bath
Dept. of Computer Science
Claverton Down, Bath, UK
jap@cs.bath.ac.uk

ABSTRACT

The functional characteristics of market-based solutions are typically best observed through the medium of simulation, data-gathering and subsequent visualization. We previously developed a simulation of multiple distributed auctions to handle resource allocation (in fact, bundles of unspecified goods) and in this paper we want to deploy an equivalent system as a distributed application. There are two notable problems with the simulation-first, application-second approach: (i) the simulation cannot reasonably take account of network effects, and (ii) how to recreate in a distributed application the characteristics demonstrated by the mechanism in the simulation. We describe: (i) the refactorings employed in the process of transforming a uni-processor lock-step simulation into a multi-processor asynchronous system, (ii) some preliminary performance indicators, and (iii) some reflections on our experience which may be useful in building MAS in general.

Categories and Subject Descriptors

C.4 [**Performance of systems**]: Performance attributes; I.2.11 [**Distributed Artificial Intelligence**]: Multi-agent systems; I.6.3 [**Simulation and Modeling**]: Applications

General Terms

Distributed systems, Refactoring, Performance

Keywords

Simulation, auctions, performance, distributed systems, multi-agent systems, AgentScape, refactoring

1. INTRODUCTION

The combinatorial auction (CA) is capable of delivering the optimal solution to a resource allocation problem. Although the theoretical cost is high, in practice, even heuristically unsophisticated solvers like CASS [5] can handle many problems quite rapidly, while CombineNet [9] services can handle a very large class of problems. However, there are circumstances that may make combinatorial auctions inappropriate: (i) if resources and bidders are distributed, the centralization intrinsic to a combinatorial auction may be problematic,(ii) under soft real-time constraints, an anytime (sub-optimal) algorithm may be preferable to an optimal algorithm with an unpredictable runtime—for example, [3] describe an anytime polynomial algorithm guaranteed to be within a bound of the optimal, and each step reduces the bound, (iii) the single-point-of-failure intrinsic to combinatorial auctions may pose an unacceptable risk for system resilience.

In an earlier paper [4], we reported on the economic characteristics and run-time performance of a market-based approach in comparison with a CA, when both are applied to common, standard data sets [5]. In this paper we present the issues that have arisen in refactoring a uni-processor agent-based simulation into a distributed agent application (using the AgentScape platform), network. There are two challenges in achieving this transition:

- **Concurrency:** how to introduce *just enough* concurrency to give an advantage but not so much as might paralyze or lead to significant numbers of delicate timing bugs. We refactor the original synchronous simulation a step at a time in order to constrain the available concurrency.
- **Communication:** distributed systems usually embody significant communication overheads. Careful placement of processes and resources helps to exploit locality. In addition, we are able to determine the impact of additional messaging overhead required to facilitate a distributed architecture.

The paper is structured as follows. The starting point is the system requirements which are set out in section 2 along with an outline of the multiple distributed auction architecture and its realization in AgentScape. We follow this with a summary of some preliminary performance data (section 3). In section 4, we reflect on what has been learnt so far and then conclude with a discussion and identification of future directions.

*This research is in part supported by the NLnet Foundation, http://www.nlnet.nl.

Cite as: Engineering Large-scale Distributed Auctions (Short Paper), Gradwell, Oey and Timmer, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal,pp. 1311-1314.

2. ARCHITECTURE

In [4] we contrast two resource allocation mechanisms:

(i) The CASS combinatorial auction solver system [5], where the solver is given a set of m indivisible non-identical items for sale and n bids and a winner determination algorithm [2] computes (NP-hard) an allocation of m items across the n bids that maximizes the total social welfare (sum of the buyer and seller profit) in the system. CASS is a freely available uni-processor application that implements the basic functionality of a branch-and-bound search.

(ii) As set of single item (one for each type of good traded) continuous double auctions (CDAs), that taken together we call a Multiple Distributed Auction (MDA). A MDA distributes the bundling task between the market traders, whose job it is to subscribe to multiple auctions to buy individual items and so satisfy bundle requests. Consequently, MDA moves the system objective from achieving the maximum valuation for the sellers (the CA objective) to maximizing the number of elements traded and achieving the best price for the buyer. We have developed a uni-processor simulation of MDA using the Repast framework [4]

These two systems have been compared using the L2 data-set from CATS [6] to produce a set of available bids and items. From a random starting position, the MDA agents (implemented using JASA [8]) then buy and sell the goods, through many rounds, until the market stabilizes and the level of trade tends to zero, indicating that none of the available items matches the bids being requested. After checking that the market has stabilized by verifying all goods have passed through the market once, the simulation then continues for as many ticks again as have already passed before we close the market.

2.1 MDA components

The Multiple Distributed Auctions system comprises:

Oracle: responsible for handing out bundles to traders on demand. Varying the Oracle's output defines the supply and demand in the market. Bundles can be both requested from the Oracle and returned to it (if they cannot be purchased/sold). All transactions are reported to the Oracle, so it maintains information and history about the market participants.

Trader: responsible for retrieving bundles from the Oracle and trading them. Traders who fail to trade their bundle within a given number of rounds must return them to the Oracle — ensuring the market does not contain too many extra-marginal traders.[1] At any one time, traders can be buyers or sellers depending on the type of bundle received from the Oracle. Traders can switch state (from buying to selling) which might happen, for example, if they decided that they could not complete a bundle as they could then sell all the items they had currently purchased.

CDA (continuous double auction): CDAs are the market places where traders can trade a single type of resource. An MDA is a collection of CDAs.

MDA Manager: responsible for telling the traders which CDAs they can use to trade the resources in their bundles. The MDA stores a reference to all CDAs and if a CDA is requested for a good type that does not already exist the MDA manager will instantiate it.

The Repast implementation of the MDA simulation uses a lock-step model, in which at every step (or round) the following three sub-steps are performed sequentially:

(i) All traders are instructed to check the status of their current bundle. If the bundle is still in progress, nothing is done. If the bundle has been completed or has failed (the trader has given up), they acquire a new bundle from the Oracle.

(ii) All auctions are instructed to perform one round. A round consists of asking all participating traders to send a shout (a bid or an ask). Any matches will be reported to the corresponding traders.

(iii) All traders are instructed to get any trade results. At this point, the statistics are updated with completed or failed bundles.

2.2 Distributed Architecture

As described above, the centralized MDA implementation runs the simulation sequentially. A single thread of execution runs the rounds of all auctions. However, in a distributed system, all processes that have no direct effect on each other can often be executed in parallel. In the case of the MDA simulation, each auction could in principle run in parallel, which could lead to performance improvements.

Unfortunately, in order for the results of the distributed MDA to be comparable to the centralized one, the notion of rounds as described above must still be kept. Within each round, we can utilize parallelism to improve performance as, *within* a step, actions can be executed in parallel. In other words, in step one, all traders can check their progress simultaneously; in step two, all auctions can run one round in parallel; and in step three, all traders can process the trade results in parallel.

The distributed implementation uses AgentScape [7], a framework for heterogeneous, mobile agents, as a base. In an agent system, typically the work is divided among several agents, which all perform a part of the work. Because AgentScape can distribute agents over multiple hosts, this helps in spreading the load of an application.

2.3 Refactoring MDA using AgentScape

The conversion of the sequential MDA simulation to one built on AgentScape takes place in three phases:

(i) **Refactoring for distribution:** The application is split into different components to be distributed by AgentScape. Preferably, each of the components should be able to function separately without requiring significant amounts of communication.

(ii) **Load balancing:** Distributing an application alone does not change the sequential nature of the application. Operations should be performed in parallel whenever this is possible, otherwise most hosts are idle most of the time. Instead, the load of each host should be maximized as much as possible, while minimizing the amount of communication that is required.

(iii) **Performance analysis:** Once the application is running on AgentScape, it is appropriate to analyze the performance, and look for hot-spots that can be optimized.

Splitting up the centralized MDA simulation into different components was straightforward, and in this case helped by the fact that the original MDA already has a modular design. Both traders and auctions (CDA) are mapped onto agents, as they can mostly function independently of each other. Each trader receives bundles of resources from the Oracle (which is also an agent) and creates a sub-trader for each one of the resources in the bundle. The task of a sub-trader is to sell/buy that particular resource in the auction that is responsible for trading that specific resource. The sub-trader uses the ZIP (Zero-Intelligence Plus) strategy [1] for bidding on the resource. For each resource, there is exactly one auction where that

[1] Buyers who have paid less than the equilibrium price and sellers who are selling for more than the equilibrium price.

resource is traded. Once a sub-trader has finished (the resource is traded or the bidding has timed out) the results are sent back to the trader that created that sub-trader. A trader has succeeded in fulfilling its bundle if all its sub-traders have succeeded in trading their respective resources from the bundle.

After distributing the simulation it is necessary to parallelize it (step (ii)). Originally, the MDA manager keeps track of the rounds and asks all of the agents in turn to perform their work. However, traders do not need to communicate with each other, and therefore, they can work in parallel (see also section 2.2). Instead of asking the traders one by one, the MDA manager can request all traders to perform their work simultaneously (using a broadcast). Subsequently, the MDA manager waits for all the traders to complete and then tells the auctions to perform their work. Similar to the traders, the auctions can also run in parallel. So, the MDA can also set the auctions running simultaneously. Once a round is finished, the MDA model enters the next round.

2.4 Performance evaluation

The initial distributed implementation in AgentScape was about 2000 times slower than the sequential implementation. After analysis, two key performance issues were apparent:

(i) The high overhead of AgentScape messaging between nodes.
(ii) The inefficiency of the sequential processing model in Repast, which meant that the truly parallel nature of AgentScape could not be fully exploited.

2.4.1 Messaging Overhead

In order to enable a smooth transition from the existing application to the distributed system, each object in the centralized application is encapsulated by an agent that permit access to these objects through remote method invocations. Consequently, these objects can now be distributed over multiple computers, but may still access each other by simply calling each other's methods.

Even though these agents help to distribute the objects, it makes the entire application significantly slower, because all method invocations are now sent over messages. In the sequential MDA, the cost of a method call was negligible, but for a distributed system, a remote method invocation has a much higher latency, due to the sending and receiving of messages. The more remote invocations, the higher the cost.

Analyzing this message traffic provided information on which agents communicated the most, and offered a first means to improve performance. Two relatively simple optimizations were as follows: (i) Cache method invocation results, whenever possible. (ii) Group method invocations that are often called in sequence into a single invocation, which saves multiple invocations. These optimizations generally do not require restructuring of the application and as such are easy to implement.

Other attempts at reducing the number of messages involved more structural changes. For example, in the MDA simulation some agents communicate only with a few others. For example, each resource (sub) trader almost exclusively communicates with a single auction object. Therefore, moving the resource trader to the node where the auction runs so that it could perform its work local to the auction saved a lot of remote messaging.

Another issue that involves a lot of messaging is polling. In the sequential MDA, traders poll the auctions for the results of their sub-traders on every round. A notification scheme, in which the auction notifies the traders only when any results are available, further decreased the number of messages.

2.4.2 Synchronization overhead

In order to ensure that the results of the centralized and decentralized systems remain directly comparable, both use a single manager for keeping track of rounds. On each round, every agent runs its part of the round, after which it waits for the next round to begin. As a consequence, in the distributed MDA a large amount of time is spent waiting. Even though auctions can run in parallel, they cannot continue working on a new round until all others are finished with the previous round.

Having to synchronize is not problematic if the costs of doing so are relatively small compared to the amount of work that has to be done by the agents. In this application, however, the average time for an auction to process shouts is less than the time it takes to send and receive a message. For example, when running an auction with a lot of traders, a message has to be sent to each one of these to get them to send in a shout.

A larger grain size would reduce the impact of the messaging overhead. This could be obtained by allowing traders and auctions to perform multiple steps at once. However, in the current design traders check the status of their bids in all of the auctions they participate in on every round. This operation allows them to maintain accurate control of funds, but it restricts the auctions and traders to performing only a single round at a time.

If the amount of work done by each individual agent is what demands the most of the processing time, then distribution would be more successful. However, in the current MDA simulation the cost comes from having to perform work for many individual agents. All agents have to be coordinated, though each individual agent performs relatively little work.

3. RESULTS AND TESTING

3.1 Testing Objectives

In our experiments, we varied two parameters. Firstly, we changed the number of servers running the AgentScape environment, to test the overheads of the inter-environment communication. Secondly, we ran multiple data-sets on both the centralized and distributed environments, to determine the run-time performance of the systems relative to each other.

Our objectives in testing were:

(i) to consider the impact of the overhead imposed by the communication between nodes, which we can demonstrate by increasing the number of nodes in the system and monitoring run-times and market throughput (the total number of bids traded in the market simulation). We would anticipate that increasing the number of processor nodes would increase the CPUs available for the system, but that adding more nodes will distribute the problem more sparsely and hence increase the amount of inter-node communication.
(ii) to run a number of simulations using the L2 data-set, to compare both the overall run-times and market throughput of the AgentScape and Repast simulations.

All our simulation experiments used a population of 400 agents.

3.2 Initial Results

Our AgentScape simulations were run on a single node and on multiple nodes on a cluster computing system. When communicating locally, the inter-agent method calls pass directly between Java objects. When the agent is remote, AgentScape uses the custom remote method invocation mechanism to facilitate communication between agents running on different AgentScape platforms.

Our experiments showed that all runs across any number of CPUs produced similar market throughput. Additionally, all experiments

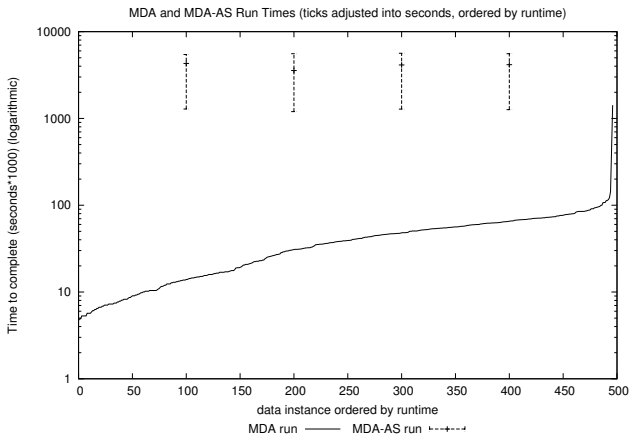

Figure 1: Run-times - MDA and MDA-AS

run to completion within a similar number of "ticks" (clock cycles). Therefore, we are reassured that our system produces identical output regardless of the number of processors used.

The major—and expected—difference between all our runs is that the run-time in AgentScape is much greater than the Repast run-time, and that the AgentScape run-time rises as the number of processors increases (as shown by the error bars in Figure 1 which give the minimum and maximum run time values, with 1 CPU being fastest and 64 CPUs being slowest). Figure 1 reproduces a graph from [4], displaying the run-time (solid line) for the different data-sets of L2, ordered by run-time, for solution by the MDA running in Repast. Above this, each error-bar shows the range and the median for solving data-sets 100, 200, 300 and 400 (as ordered by run-time) from L2 using 1, 2, 4, 8, 16, 32 and 64 processors. Predictably, the bottom of the error-bar corresponds to 1 processor and the top to 64.

4. EVALUATION AND DISCUSSION

MAS software development is characteristically evolutionary and a common starting point is a proof-of-concept system running on a single machine utilizing an agent platform or even a simulation framework. The software engineering challenge lies in how to scale that demonstrator up into a system comprising many more agents running over multiple machines. A large-budget solution might throw away the demonstrator and rewrite from the ground up, but there are many risks with this approach as well as high costs. An alternative approach, that is the subject of this paper, is to refactor the demonstrator into a large-scale system, taking advantage of the incremental nature of the changes applied and regression testing to create confidence in the process and the outcome.

We have pursued this approach and have transformed a centralized Repast simulation of a Multiple Distributed Auction (MDA) into a multi-agent system, supporting large numbers of agents participating in large numbers of auctions on distributed machines. The AgentScape mobile agent platform was used to distribute the entities in the MDA over multiple hosts and to provide the necessary communication between these entities. Unfortunately, the empirical evidence from the experiments show that the communication overhead of the distributed market is quite large compared to the benefit of gaining more computing resources.

As discussed, the large communication overhead is mainly due to encapsulating all the messaging in an RMI data-exchange mechanism. In fact, the grain-size of the work that can be done in parallel is relatively small compared to the communication overhead. In addition, the problem size (1000 goods, 256 bids, 400 agents) or "payload" is sufficiently small to be computable on a single high performance CPU.

However, the choice to constrain the asynchronous nature of the distributed simulation by keeping the notion of rounds, did make it possible to compare the results with the centralized simulation. Unfortunately, this choice also meant much synchronization between traders and auctions, which lowered overall performance. A fully asynchronous simulation would probably show better performance, and this is the objective of the next stage of our work.

In conclusion, it has proved to be difficult to optimize the speed of the MDA market simulation by distributing over multiple hosts, while still respecting the synchronization constraints from the original Repast simulation. Synchronizing agents is very time consuming due to the amount of messaging involved. However, the barrier synchronization can be removed, at the cost of a more complex refactoring of the original code, and at the cost of producing *similar* but not identical results to the original code. Thus, the process reported here has been a tedious but necessary step to demonstrate functional *equivalence* before we move on to a situation in which we must define and demonstrate functional *similarity*.

5. REFERENCES

[1] D. Cliff. Minimal-intelligence agents for bargaining behaviors in market-based environments. Technical Report HP–97–91, Hewlett Packard Laboratories, Bristol, England, 1997.

[2] P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial Auctions*. MIT Press, 2005. ISBN: 0-262-03342-9.

[3] V. D. Dang and N. R. Jennings. Optimal clearing algorithms for multi-unit single-item and multi-unit combinatorial auctions with demand/supply function bidding. In *ICEC '03: Proceedings of the 5th international conference on Electronic commerce*, pages 25–30. ACM Press, 2003.

[4] P. Gradwell and J. Padget. A comparison of distributed and centralised agent based bundling systems. In *ICEC '07: Proceedings of the ninth international conference on Electronic commerce*, pages 25–34, New York, NY, USA, 2007. ACM Press.

[5] K. Leyton-Brown, E. Nudelman, and Y. Shoham. *Empirical Hardness Models for Combinatorial Auctions*, chapter 19. MIT Press, 2006.

[6] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, 2000.

[7] B. J. Overeinder and F. M. T. Brazier. Scalable middleware environment for agent-based Internet applications. In *Applied Parallel Computing*, volume 3732 of *Lecture Notes in Computer Science*, pages 675–679. Springer, Berlin, 2006.

[8] S. Phelps, M. Marcinkiewicz, and S. Parsons. A novel method for automatic strategy acquisition in n-player non-zero-sum games. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 705–712, New York, NY, USA, 2006. ACM Press.

[9] T. Sandholm. Expressive commerce and its application to sourcing: how we conducted $35 billion of generalized combinatorial auctions. In *ICEC '07: Proceedings of the ninth international conference on Electronic commerce*, pages 349–350, New York, NY, USA, 2007. ACM Press.