Leasing in a Market for Computing Capacity

Spyros Lalis and Alexandros Karipidis

Computer Science Dept.,
University of Crete, Hellas
{lalis,karipid}@csd.uoc.gr
Institute of Computer Science,
Foundation for Research and Technology, Hellas
{lalis,karipid}@ics.forth.gr

Abstract. One of the challenges in large scale distributed computing is to utilize the thousands of idle personal computers connected to the Internet. In this paper, we present a system that enables users to effortlessly and safely export their machines in a global market of processing capacity. Efficient resource allocation is performed based on statistical machine profiles and leases are used to promote dynamic task placement. We show how leasing, as opposed to static resource allocation, yields a natural renegotiation process through which prioritized computations receive better quality of service and resource providers enjoy bigger profits.

1 Introduction

The growth of the Internet has provided us with the largest network of interconnected computers in history. Many of these systems are often under-utilized, a fact accentuated by the globe's geography since "busy" hours in one time-zone tend to be "idle" hours in another. Distributing computations over the Internet is thus very appealing.

Several issues must be resolved for this to be feasible. Platform heterogeneity must be overcome and security problems arising from the execution of code from untrusted parties must be confronted. In order to register an available machine as part of a computing infrastructure, a corresponding runtime environment must also be downloaded and installed. This not only costs time but also introduces several administration headaches, practically limiting current resource sharing systems. Moreover, in most cases, little is done to make participation attractive to third parties, which we believe is of key importance if such a system is to be widely used.

In this paper we present a platform that addresses these problems, promoting distributed computing over the Internet considerably. In this system, hosts advertise their intention to serve as resource providers in a global market. Clients enter this market to locate machines that are appropriate for their computation. Matchmaking between supply and demand occurs via an economy-based algorithm producing contracts. Contracts have an expiration time, thus are essentially leases of use. In the case that resources are charged for, leasing guarantees

C. Linnhoff-Popien and H.-G. Hegering (Eds.): USM 2000, LNCS 1890, pp. 318–325, 2000.

that profitable contracts will be assigned to resource providers without any intervention on their behalf. With leases it is also possible to place an upper limit to the amount of time prioritized computations can be blocked or experience performance degradation due to other computations with less priority.

2 System Architecture

In order to guarantee cross-platform operability, yet at the same time minimize administration overhead and achieve execution safety for hosts, the system is implemented in Java. Providers connect to the system by pointing a web browser to a given address from which the runtime system is automatically downloaded to their machines as a Java applet. Client applications connect to the system via an application programming interface (API).

Both providers and clients submit orders to a server, specifying their actual and desired machine profile respectively. Hence, they act as sellers and buyers in a market of computing capacity. Given the fact that any machine connected to the Internet can place a sell order, and anyone can use the system API to write applications issuing respective buy orders, a universal market is formed.

An overview of the system's architecture is depicted in Fig. 1. The basic system components are the market server, hosts, the host agent, schedulers, tasks and client applications (or front-ends). The role of and the interactions between these components are briefly discussed in the next subsections.

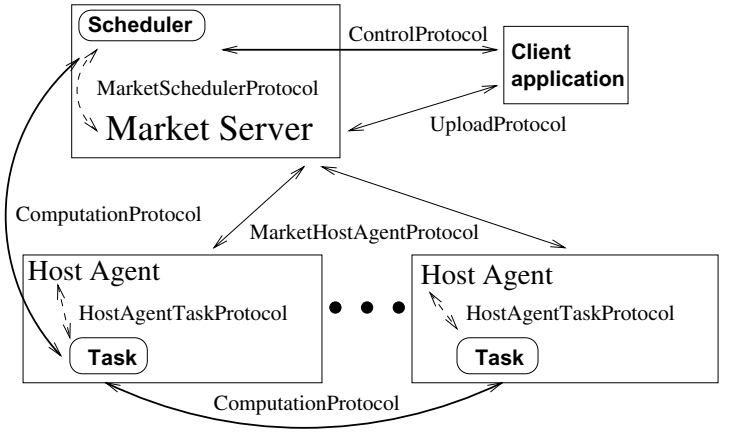

Fig. 1. Overview of architecture

The *Client Application* is a program which needs to perform computations that require considerable processing power. Through the system, it may either distribute a computation across a number of machines or just delegate the execution of an entire computation to a fast idle machine to speed up execution.

The *Market Server* is the meeting place for buyers and sellers of processing power. It collects orders from clients and hosts. Using this information, it then matches buy with sell orders and thus allocates resources.

A *Host* is a machine made available to be used by clients. A host participates in the market through the *Host Agent*, a Java applet. The user visits a URL with a Java enabled web browser and the agent is downloaded to her system. The agent communicates with the market server, takes care of placing orders on behalf of the user and executes tasks assigned to the host. It also provides the market server with the benchmark scores needed for the host's profile.

Computations consist of a *Scheduler* and one or more *Tasks*. The scheduler runs on the market server, placing orders in the market for acquiring machines. New orders can be issued at any time to adapt to changing application requirements or market conditions. When a host is allocated to the scheduler, a task is launched in that machine to assist in completing the computation.

3 Resource Allocation

Host allocation is based on machine profiles. Both hosts (sellers) and clients (buyers) submit orders to the market server, specifying their actual and desired machine profile respectively. The profiles include the mean and variance for a set of benchmarks over key performance characteristics such as integer and floating point arithmetic. Part of a profile is also the host *abort ratio*, which is the ratio of computations killed versus computations initiated on that host (a "kill" is caused when a host abruptly leaves the system in the midst of an ongoing computation). The performance vectors and abort ratio of host machines are automatically produced by the host agents. Profiles can be easily extended to include additional information that could be of importance for host selection.

Further, a credit based [1] mechanism is used for charging. Credit can be translated into anything that makes sense in the context where the system is deployed. Within a non-profit institution, it may represent time units to facilitate quotas or to introduce priorities. Service-oriented organizations could charge clients for using hosts by converting credit to actual currency.

An economy-based mechanism is employed to match the orders issued by providers and application clients. For each match, the market produces a lease, which is a contract between a host and a client containing their respective orders and the price of use agreed upon. Leases are produced using continuous double auction [6]. A lease entitles the client to utilize the host for a limited amount of time. If the client's task completes within the lease duration, then the buyer transfers an amount of credit to the seller as a reward, calculated by multiplying actual duration with the lease's price per second. If the lease duration is not honored, an amount of credit is transfered from the dishonoring party to the other as a compensation.

Since leases have an expiration date, it may not be possible to maintain the same set of hosts allocated to a computation, for its entire duration. In fact, this is highly unlikely to happen if the computation is lengthy, compared to

the lease durations specified by the corresponding host providers. Moreover, at some points in time, an application may not be able to get any hosts at all to perform its tasks. As an example, Fig. 2 shows a trading scenario where an application eventually fails to keep the same host due to a price raise. An important consequence is that application schedulers must be able to also deal with (temporary) unavailability of hosts.

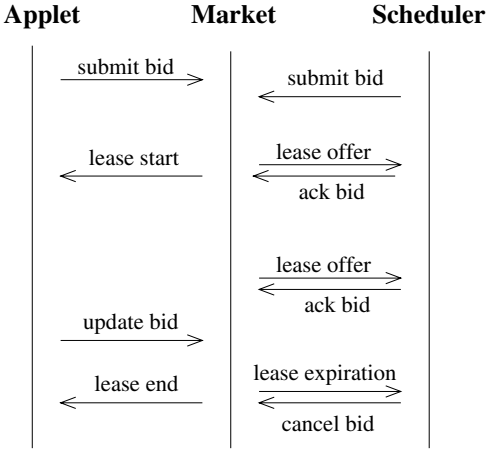

Fig. 2. A trading scenario

The trading protocol is designed to minimize communication between hosts and the market (the number of hosts is expected to be very large). Hosts contact the market only to submit/change their bids while the market communicates with hosts only when their current lease status changes. The interaction between the market and application schedulers is more tight in order to allow for flexible and timely resource negotiation. Essentially, the market notifies schedulers each time a bid can be converted into a contract as well as each time a contract is about to expire. In the former case, the scheduler may either accept or turn down the contract. In the latter case the scheduler may withdraw its bid (as in Fig. 2), leave it unchanged, or update its bid to actively try to re-gain possession of a host. Schedulers may also submit new bids at any point in time.

4 Leasing vs Buying Computing Resources

To illustrate the importance of leasing in an open environment, such as the Internet, we present an experiment conducted using a prototype version of the system. These measurements show how renegotiation affects both the quality of service experienced by clients and the amount of credit collected by providers.

The scenario of the experiment for which the measurements were made is as follows. We let two identical and resource intensive computations, C1 and C2,

enter the system under conditions of resource scarcity, i.e. there is not enough capacity to accommodate both computations at the same time. These computations are assigned a different budget for performing their calculations; let Budget(C1) << Budget(C2). Finally, the computation with the lower budget (C1) is given a head start of a few seconds, followed by the other computation (C2). The same scenario is executed twice, one time for an infinite and the other for a limited leasing duration respectively. Leases with no expiration date lead to static resource allocation, meaning that once a resource is allocated to a computation it will not be freed unless the computation terminates.

As it can be inferred from Fig. 3 static resource allocation has a particularly disturbing effect. Since C1 enters the system first, it acquires all resources, and continues to hold them even in the presence of C2, which has a bigger budget. Assuming that budget reflects priority, this implies that a prioritized process is actually blocked by another process of lower priority. Notice that blocking occurs despite the fact that resources are auctioned. While auction mechanisms guarantee price competition they require *all* competitors to be present when the auction takes place. This is obviously impossible in an open system where clients and providers may enter and leave anytime.

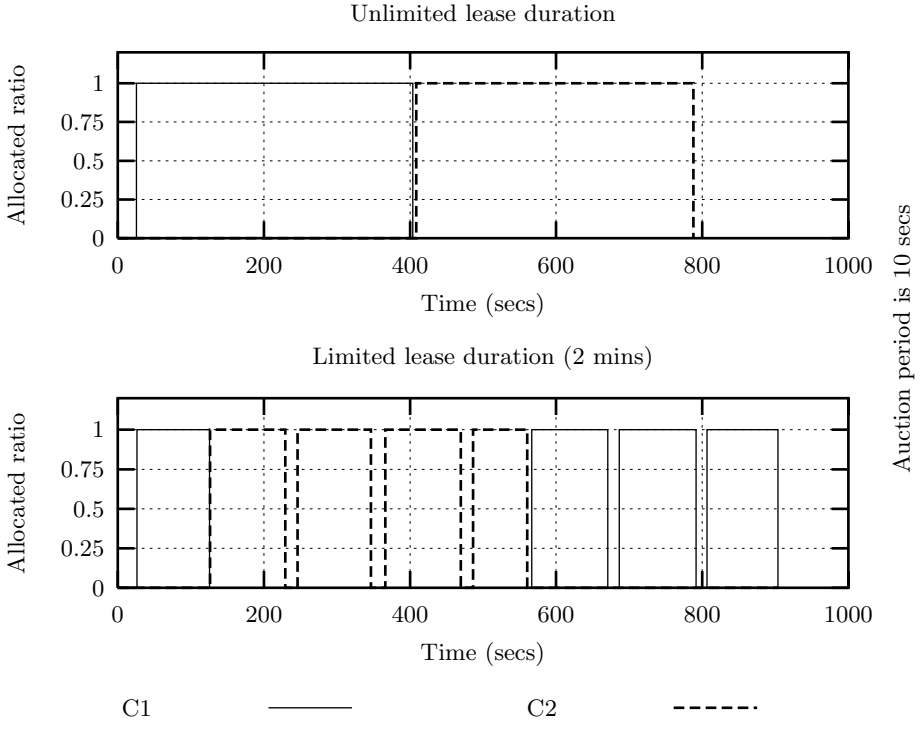

Fig. 3. Allocated resources

This effect is largely eliminated case when resources are leased for a reasonably limited duration. In this case, even though C1 seizes all resources upon entry, these are re-allocated to C2 when the leases expire. Therefore, the total execution time of C2 is only slightly longer than the actual time spent for performing the computation. This is a noticeable difference compared to the first case. Conversely, only when C2 terminates, can C1 reclaim these machines and complete its execution. It can thus be said that leasing compensates for the impossibility of predicting future system traffic in a dynamic system. In other words, leasing is the analog of timesharing within a multiprocess system.

It is worthwhile pointing out another implication of leasing. Even though resource utilization is the same for both cases, providers collect more credits under leasing, as shown in Fig. 4. This is particularly important in a real-life situation were resources are unlikely to be given for free and providers wish to recuperate their costs as quickly as possible. Notably, the increase in provider income does not come at the cost of the low budget computation C1, which allocates resources at the same price yet at a later point in time. It is the computation with the high budget that "pays" more, but always according to its true valuation.

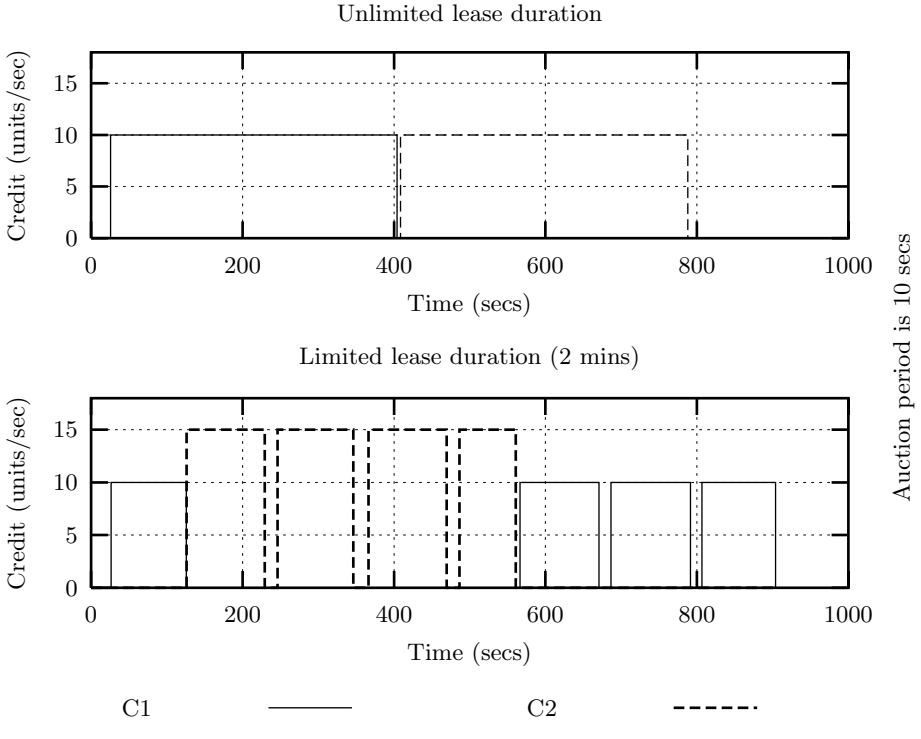

Fig. 4. Provider income

Leases are also practical in cooperative environments. The lease duration allows users to indicate when their hosts are under-utilized. Based on this knowledge, tasks can be placed on hosts that will be idle for enough time, and checkpoints can be accurately scheduled, right before a host becomes unavailable.

5 Related Work

Legion [7], Globus [5] and Condor [8] have mechanisms for describing resource properties and performing matching. These mechanisms were created in order to make it possible to respect local access control policies of hosts and were not oriented towards a market-based approach. None of them uses auctions in order to perform matching of computations to hosts and as a consequence, in order to maximize income in a real market situation, providers would have to constantly monitor the supply and demand for resources and change their offers correspondingly. These systems are also architecture-specific which constrains the market's size and partitions competition. Finally, all require extensive administration which prohibits a low overhead participation in the market.

Other systems using Java have been designed for supporting distributed computations. Charlotte [3], automatically distributes computations over machines. However, it does not employ market-based principles to allocate hosts. The market paradigm has received considerable attention in distributed systems aiming for and efficient resource allocation [4]. Popcorn [9] also uses auction mechanisms but does not provide leasing, which as demonstrated above is required in order to support a dynamic market. The Java Market [2] uses the framework described in [1]. However, it does not use auctions, leasing or an interactive tool for providers to specify and update the price for their machines. It is also not suitable for interactive applications, as source code must be compiled on the server and results are e-mailed back to the client. Therefore it is hard to provide market services in the form of a coherent, user-friendly application which can be installed on a client's desktop.

Currently, no other system offers ease of participation, allocation decisions assistance, economic efficiency and periodic renegotiation. We provide ease of participation through an effortless web interface for sellers. Buyers also participate easily by simply running market-aware applications as they would any other application. Economic efficiency is achieved using auctions whereas allocation decisions are assisted through host profiling. Lastly, renegotiation is enforced and – in combination with the auction mechanism – provides prioritization over a network of heterogenous runtime environments.

6 Future Directions

An issue that we wish to investigate is how the cost of check-pointing (and migrating between hosts) is weighed against the cost of keeping tasks on hosts

with degraded performance. It would be interesting to contrast the performance of leasing with checkpoints versus static allocation without checkpoints.

Moreover, we wish to experiment with schedulers capable of recording the performance of previous allocations. Such schedulers can be regarded as the "computational" counterparts of information agents that learn the users' preferences and scan the web for corresponding documents. In our case accumulated information can perhaps be converted into "experience", leading towards more adaptive resource allocation strategies responding to the different requirements of individual applications.

Lastly we would like to look into trading and scalability issues. We are currently considering improvements to the way contract acknowledgment and expiration is handled in order to support a more flexible and efficient re-negotiation process between the market and application schedulers. Also, the current architecture is limited by the market server. A single server could not handle hosts connecting to a truly world-wide version of this service. We intend to overcome this problem by introducing multiple market servers that will allow traffic to be shared among several geographically distributed servers.

References

[1] Y. Amir, B. Awerbuch, and R. S. Borgstrom. A cost-benefit framework for online management of a metacomputing system. In *Proceedings of the First International Conference on Information and Computation Economies*, pages 140–147, October 1998.

[2] Y. Amir, B. Awerbuch, and R. S. Borgstrom. The java market: Transforming the internet into a metacomputer. Technical report, Johns Hopkins University, Center for Networking and Distributed Systems, 1998.

[3] A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the web. In *Ninth International Conference on Parallel and Distributed Computing Systems*, September 1996.

[4] S. H. Clearwater, editor. *Market-based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1995.

[5] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2), 1997.

[6] D. Friedman. The double auction market institution: A survey. In D. Friedman and J. Rust, editors, *Proceedings of the Workshop in Double Auction Markets, Theories and Evidence*, June 1991.

[7] A. S. Grimshaw and W. A. Wulf. The legion vision of a worldwide computer. *CACM*, 40(1):39–45, 1997.

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

[9] O. Regev and N. Nisan. The POPCORN Market – an Online Market for Computational Resources. In *Proceedings of the First International Conference on Information and Computation Economies*, pages 148–157, October 1998.