The Java Market: Transforming the Internet into a Metacomputer

Yair Amir[1], Baruch Awerbuch, and Ryan S. Borgstrom[2]
Department of Computer Science
The Johns Hopkins University
yairamir, baruch, rsean@cs.jhu.edu

Abstract.

Most of the machines that are connected to the Internet are idle a significant fraction of the time. This paper presents the *Java Market*, a system that allows organizations and Internet users to make use of this wasted computational power.

Using the Java programming language and the Web technology, the Java Market is the first metacomputing system that can seamlessly take advantage of machines of any architecture, anywhere on the Internet. Every user on the Internet can contribute their machine's computational resources just by pointing a Java-capable browser to the Java Market web page. Similarly, users can launch jobs to the system by posting them on the Web and registering them with the Java Market.

In contrast, other systems that allow sharing of computational resources between machines over the network require homogeneous system architecture. They often involve extensive installation or even kernel-level modifications to the operating system.

1. Introduction

The wasted CPU cycles on idle machines can more profitably be put to use solving problems for other users. The key technology for this is the ability to transfer jobs from one machine to another. Traditionally, implementing this technology is difficult, even using one of the software packages available, and it becomes much harder on systems of heterogeneous machines.

The goal of the Java Market, presented by Figure 1.1, is to make it possible to transfer jobs to *any* machine on the Internet that desires to participate, without any installation or platform-dependent code. To accomplish this, we use the recently developed web technology and the Java Applet architecture. Conceptually, the Java Market transfers regular Java applications submitted to it by a consumer into a Web browser running on a producer's machine. The Java Market modifies the submitted application. It transforms it into an applet using special market classes for file I/O, and modifies the code for enhanced Market control and security. The Java Market is, to our knowledge, unique in that it can distribute ordinary Java applications – the programmer does not have to use an unusual programming model in order to take advantage of the Java Market's power. Similarly, companies and individuals will not have to redesign existing code in order to use it with the Java Market.

The Java Market approach has several notable advantages. The Java Market has no dependence on any given architecture – both producers and consumers can be running

[1] Yair Amir is also at the Center of Excellence in Space Data and Information Sciences, NASA.

[2] This research is supported, in part, by the Defense Advanced Research Projects Agency (DARPA) under grant F306029610293.

any kind of machine, and any operating system, that has a Java-capable Web browser. In other words, it can handle heterogeneous machines with no modifications. For the same reason, the Java Market can manage machines anywhere on the Internet, as long as it can establish a network connection to them.

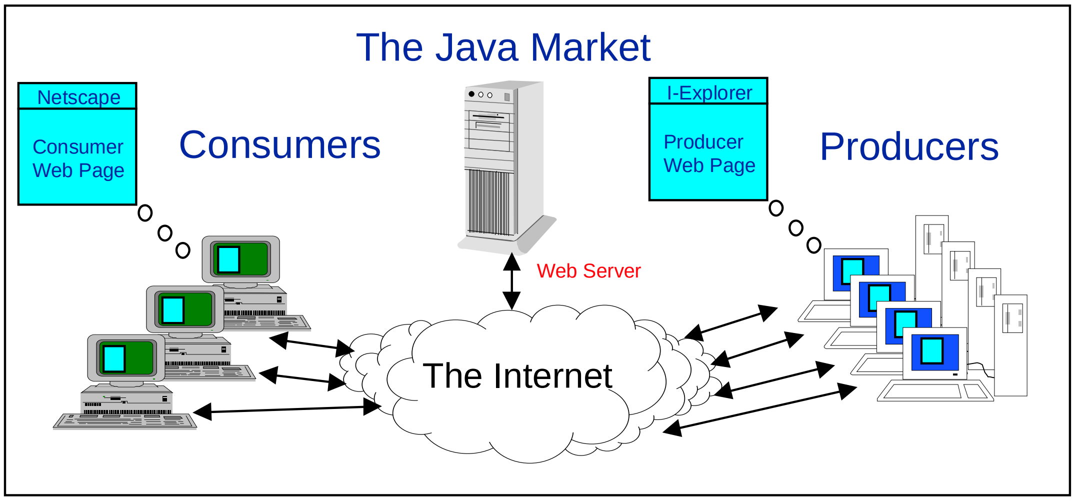

Figure 1.1: An Internet-Wide Metacomputing System.

Producers and consumers can use the Java Market with minimal effort. The only programs that they need to run are secure *applets*, which can be executed automatically by their Web browsers when they connect to the Market web pages. Even the *application* that runs on the Market's home machine and manages all of the resources and tasks is completely portable; it is written in Java, and all of its system-dependent features are set by command-line options. Its built-in economic framework makes it easy to create limited Markets for exchanging computational resources between specific companies (using real money) or within a single organization (using virtual money). Leasing out your machine via the Market web pages is simple enough, in either case, that it can be thought of as a kind of screen saver. Instead of turning on a fractal pattern when you leave for the night, a browser window can automatically be opened to enable the Java Market to use that machine for computation.

Java is quickly becoming a common language, and this framework uses it to its best advantage. There are some drawbacks, however. Because jobs are executed within a browser window, they must be written in Java. (This is an important limitation now, but as Java becomes more popular, it becomes less of a limitation). It is difficult to do automatic checkpointing, although users can do it themselves if the job is very long and vitally important. Finally, the Market must be able to read the source code for the jobs, which requires that that source code be Web-readable (at least, to the Market machine). In principle, the Java Market could read Java byte codes from the consumer instead of the source, to allow better privacy. This is not yet implemented. It is important to note that only byte codes are shipped to producer machines.

In addition to the concepts and advantages listed above, the Java Market uses several unique strategies to help it perform admission control and resource allocation. In a context where every job has a *reward* associated with it, and every producer has a *cost*, using either virtual or real money, these strategies will provide competitive profits and a measure of security against system failures.

Related Work

The Mosix system [BGW93] provides a set of enhancements to Unix which incorporate the CPU resources of a Network of Workstations by enabling preemptive migration of processes and dynamic load balancing within a network that has installed Mosix. These enhancements are completely transparent to the application level. Mosix is limited to BSDI Unix and the x86 Intel architecture.

The Condor system [PL95] is similar to our Java Market framework. The Condor team has created a set of software tools for utilizing the wasted CPU cycles in a cluster of workstations. Condor provides a checkpointing mechanism for the jobs it schedules, which allows it to save the state of a job when an owner returns to a workstation and continue it later when another machine (of the same architecture) becomes available, ensuring progress.

The Globus project [FK97] defines an extensive framework for metacomputing. The project builds a metacomputing toolkit that encapsulates a collection of low-level services (for resource allocation, communication, authentication and file access). On top of these, high level services such as resource brokers, directory servers, and schedulers can be built. This general framework will allow integration and interoperability of different (and geographically distributed) computer systems that will be built according to this framework.

The Popcorn project [CLNR97], independently developed at the Hebrew University in Israel, shares many of the Java Market’s features. It can distribute the subroutines of a given program, but it requires that all users write their Java programs using the Popcorn model. This means that the Popcorn market cannot be used with standard Java applications.

The Milan project [BKKK98], like the Popcorn project, provides a programming model that can take advantage of heterogeneous Internet-connected machines. Also like the Popcorn project, Milan is not designed for use with standard Java applications.

The Piranha project [CFGK94] is a system for parallel computation that can also take advantage of the Java language. Piranha programs are written using a new "adaptive parallelism" paradigm that can easily distribute pieces of a job to available machines and remove them from newly-unavailable machines. Adaptive parallelism is an entirely new style of programming, with its own benefits and drawbacks, and as such is not directly comparable to the Java Market; however, it provides an interesting direction for research into using idle cycles on workstations.

The Java programming language [Java, DD97] plays a crucial role in our work. Java is the first major language to provide a full implementation of the applet concept. An applet is a program, stored on the World Wide Web, that is run by a Web browser, on the Web browser's machine.

Java programs do not need to be edited to run on different machines; instead, they are written with a *virtual machine* in mind. The machine that it actually runs on translates Java's commands into their equivalents on that machine. This feature means that the creator of a Java program need not care which machine his code is actually run on; it executes the same way anywhere, although at different speeds.

Java was designed with a high level of security, so there is no risk to the user of a browser when they run a Java applet. This security extends from control over file access to limits on the net connection ability of an applet.

2. The Model

There is a widespread shortage of computational resources. Workstation users often have limited access to other machines, and all their jobs must share a single CPU. At the same time, there is an incredible surplus of computational resources. For every user whose machine is overloaded, there is another who is not using their machine at all. The problem addressed in this paper is the proper distribution of surplus computational resources to those who have a temporary shortage. An ideal solution would turn a LAN, a WAN, or the Internet itself into a single *metacomputer* or metacomputing system -- a virtual machine that harnesses the combined power of its components as if they formed a single strong machine.

A *task* is a job submitted to the metacomputing system by a *consumer*. A task has the following properties associated with it:

- A program and its data input.
- A reward (to the metacomputer) for its timely completion.
- The number of cycles required to complete the task.
- A deadline by which the market must complete the task to win the reward.

A *producer* is a machine leased to the metacomputing system that has the following properties associated with it:

- The cost of using that machine.
- The number of computational cycles per second the machine can execute.
- The cost of I/O using that machine.

The *market* is a mediator between consumers and producers. The market assigns tasks to producer machines according to *resource allocation* algorithms. The market goal is to maximize its profit, which is the total reward minus the total cost of resources used.

It is natural to associate real-world money with these costs and rewards. However, a market can also be implemented with virtual money, which then serves as a tool to increase the overall market efficiency. For example, this environment may be used with real-world money to lease computational resources over the Internet. It can also be used within organizations with virtual money in order to make use of otherwise wasted cycles.

We can evaluate a metacomputing environment based on its performance in the following areas:

1. The guarantees it provides regarding the level of service tasks receive.
2. Its awareness of changing resource availability.
3. Its security against unfair use and malicious abuse.
4. Its ability to accommodate resource heterogeneity.
5. Its ability to scale to large systems.
6. Its ability to quantify the power of the available resources.
7. Its ability to impose a desirable scheduling policy.
8. Its transparency, where transparency is the user's lack of awareness that the metacomputer is not a single machine.

The rest of this paper describes a metacomputing system working within this model and evaluates it according to the measures for performance given above.

3. Properties of the Java Market

This section describes the properties of the Java Market as they relate to the above metacomputing environment considerations.

3.1. Service Guarantees

The Java Market is not intended to have complete control over the producer machines. Instead, it is optimized for security, using only the machine's computational and networking resources. Producer machines can be removed from the market at any time, either by network partition, a machine crash, or user intervention.

The future availability of a machine must be extrapolated from its past availability, and the correlation between past and future availability may be hard to determine. The obvious way to deal with failures and machines ceasing to be available is by assigning jobs to more than one machine. The drawback of this method is loss of resources. Another approach is to try to choose a machine that will have a good chance of staying operational for the required duration of time. Intuitively, it seems that the best strategy is choosing a machine that has stayed reliable for the longest period of time. However, in an adversarial scenario, it is always possible to defeat such protocol by killing the longest operating machine.

Fortunately, the *Winner Picking* strategy [AAFL96] can give us a guaranteed high probability that a job will finish on time, if there is at least one producer machine available for a significant length of time, regardless of the correlation between past and future. The Winner Picking strategy assigns a job to each machine with a probability that grows exponentially with each unit of time that the machine is continuously connected to the Market. As a consequence, long-term jobs will be executed on machines that have been connected for a long time, and short-term jobs can be executed on relatively flaky machines. At the same time, the randomized nature of the Winner Picking strategy can defeat an adversarial scenario.

Even in a hostile environment, where the Java Market is given very limited rights to the producer machines, the Winner Picking strategy attempts to provide a degree of Quality of Service. The Market will restart tasks that were running on crashed producers, choosing a faster machine if possible. To minimize user intervention, it allows producer machines to specify the length of time they can guarantee availability in exchange for a greater reward.

3.2. Awareness of Changing Resource Availability

The Java Market remains in continuous contact with producers. When a producer disconnects or crashes, the Market immediately detects the loss of this resource and removes it from its list of producers.

3.3. Security

The Java Market's basic design comes with certain security guarantees. First, the Java Market guarantees a level of security equal to or better than the security of the web browser the producer uses. All tasks are run as ordinary Java applets, with the limits imposed by the "Java sandbox" paradigm – even though they are written as regular Java applications. As long as the sandbox in the producer's browser meets the Java specifications, the Market can guarantee that the producer's system will not be damaged or rendered unusable.

However, the security provided by the Java sandbox is not sufficient. For example, the sandbox does not prevent an applet from taking control of the producer's CPU resources. It would normally be perfectly legal for an applet to break the Market's paradigm and continue executing within the producer's virtual machine indefinitely. The Java Market automatically modifies the consumer's Java code. These automatic, code-level, modifications protect the market and the producer alike from such abuses and other misuses of the system.

3.4. Heterogeneity of Participating Machines

The Java Market does not run on any of the standard machine architectures. Instead, it executes tasks on the Java virtual machine. This virtual machine is an integral part of all Java-capable Web browsers, which are available on the vast majority of machines and operating system architectures.

3.5. Scalability

The Java Market can handle machines and task requests from anywhere on the Internet. Experience with the Condor system [PL95] shows that one centralized scheduler can manage hundreds of producer machines as long as the scheduler acts only as a decision-maker.

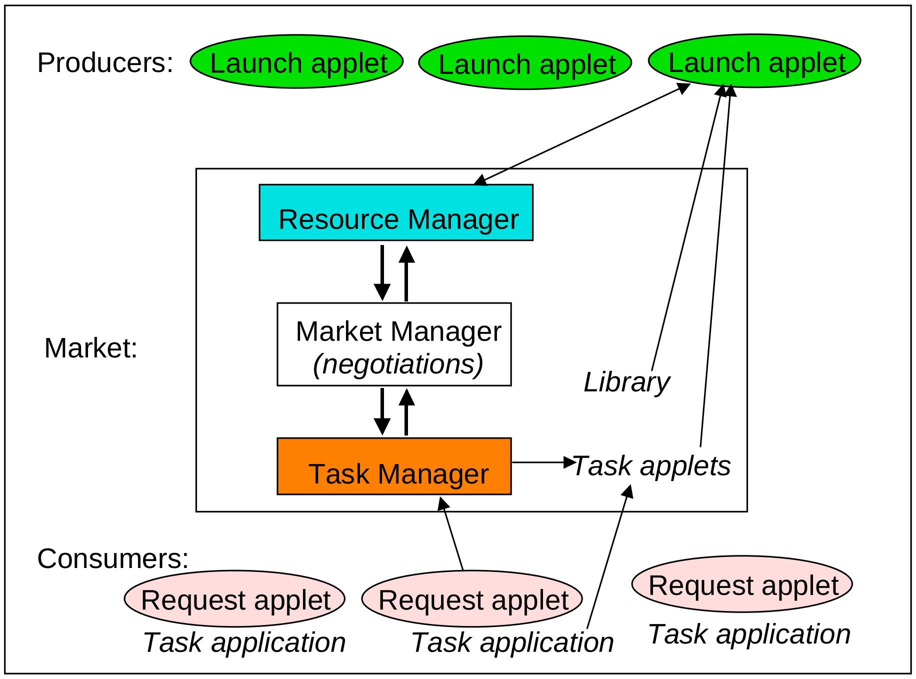

Figure 4.1: The Java Market Architecture.

3.6. Quantifying the Power of Producer Machines

Producers of computational resources access the Java Market via a Web page. This imposes some fundamental limits on what the Market can quantify about the producer machine. Web-executable Java applets are forbidden to perform many kinds of system calls and system probes. Two things the Java Market can discover empirically are CPU speed and the speed of the network connection between the Market and the producer. It does so.

3.7. Scheduling Policy

The Java Market uses the unique *Cost-Benefit* framework being developed at the Johns Hopkins University. In effect, it "pays" producers based on the quality of service they can provide, "charges" consumers based on their consumption of resources, and attempts to maximize its "profit". This is in contrast to most fair scheduling policies that try to maximize the utilization of the available resources over time.

3.8. Transparency

The Java Market is not intended to be a transparent metacomputer; instead, our goal is *ease of use*. Users need to explicitly invoke the metacomputing power of the Java Market, but this is not significantly more difficult than using their own machine. People who wish to offer their machine's services simply connect to a page on the Market's Web site. Submitting a task to the Market (for execution on some appropriate machine or machines) is the same process, although the task must be an application written in Java. Currently, results are sent to the user as email.

4. General Architecture of the Java Market

The Java Market uses the Web and the Java language as its primary tools. A producer makes their machine available as a resource by directing its browser to one of the Market web pages. A consumer registers its request for computational resources by contacting another of the Market's web pages, and posting its program (written in Java) in a Web-accessible location. The Java Market is a metacomputing system that performs admission control for tasks, signs contracts with and manages producer machines, and places tasks on producer machines based on advanced resource allocation algorithms.

The Java Market is composed of three main entities, as depicted in Figure 4.1:

- **The Resource Manager** keeps track of the available machines – those that have registered themselves as producers.
- **The Task Manager** keeps track of the consumer-submitted tasks.
- **The Market Manager** mediates between the Resource Manager and the Task Manager.

4.1. The Resource Manager

The Resource Manager is run on the Market machine. When a producer registers with the Market web pages, they automatically run a special applet, the *Launch Applet,* that tells the Resource Manager about the producer's machine's power. The Resource Manager stores this information, as well as the state of the machine (in this case, 'available,') the IP address, and so on. This information forms a machine profile.

The Launch Applet is the part of the Resource Manager executed on the producer's machine. This gives it two responsibilities. The first is resource discovery, that is, assessing the machine's computational and networking capabilities. The second is directing the producer's browser to a page containing whatever task is assigned to it.

4.2. The Task Manager

The Task Manager is run on the Market machine. When a consumer registers their task with the Market web pages, they run another special applet, the *Request Applet,* that gathers information about the task they want the Market to perform. Once this information is gathered, it is sent to the Task Manager. The Task Manager then gathers all the Java files and input files associated with the task from the Web, edits the Java files as necessary, compiles them, and then passes the entire task to the Market Manager.

The Request applet is the part of the Task Manager that is executed on the consumer's machine. Its primary responsibility is to wait. First, it waits while the consumer types in the necessary data about their task. It sends this information to the Task Manager proper, and then waits again. As it waits, the Task Manager downloads, edits, and compiles all of the Java code associated with the task. When this is complete, it tells the Request Applet that the task has been accepted or rejected by the Market. The Request Applet displays this information and ceases computation.

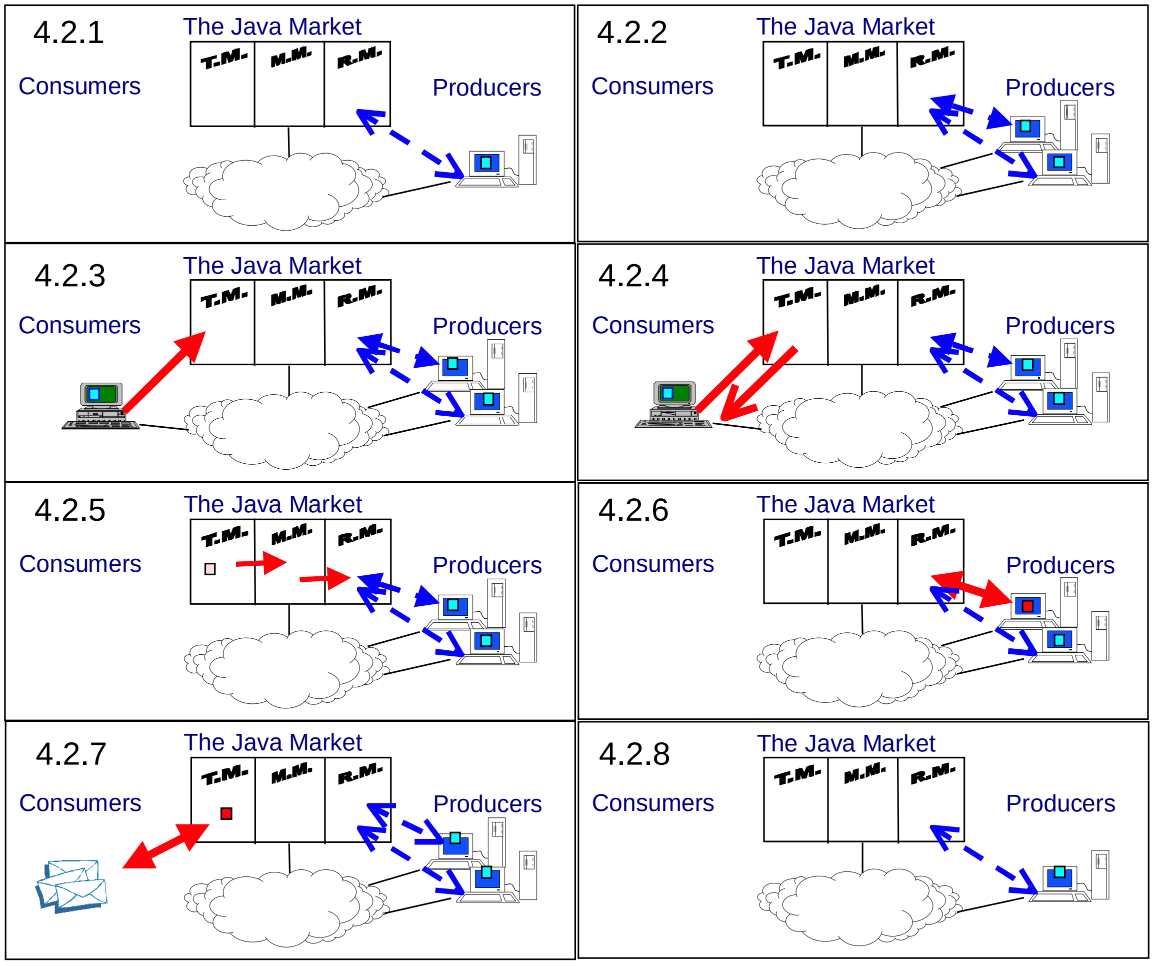

Figure 4.2: An Example Scenario.

4.3. The Market Manager

The Market Manager acts as an overseer, performing resource allocation and admission control. There are two issues that must be addressed: the uncertain *availability* of a producer machine for the length of time a given task requires, and the on-line decision-making necessary to maximize the profit of the market. There are two theoretical tools that the Market uses to make these decisions: the *Winner Picking* strategy [AAFL96] and the *Cost-Benefit* framework [AAP93].

Our admission control algorithm is based on the *Cost-Benefit* framework. The central concept of this framework is the *opportunity cost* of a resource. This opportunity cost is a cost function that increases exponentially with a resource's utilization. A task will be accepted or rejected based on whether the reward it offers is larger than the opportunity cost of executing that task on some sufficiently fast producer machine.

Once a task has been accepted, we allocate a producer to that task using a combination of the Cost-Benefit framework and the *Winner Picking* strategy, previously explained. It is proven in [AAFL96] that an appropriate producer for a task will be selected after at most a small number of failed assignments.

4.4. An Example Scenario

The scenario presented in Figure 4.2 demonstrates how the Java Market metacomputing system works.

1. A producer machine registers its availability over the network with the Java Market.
2. A second producer machine registers its availability with the Java Market.
3. A consumer connects to the Java Market and registers a task.
4. The Task Manager downloads the task information from the Web and modifies and compiles the code. It then notifies the consumer of the consumer's success at launching the task.
5. The Market Manager mediates between the Task Manager and the Resource Manager in order to find an appropriate producer to execute the task.
6. The selected producer's browser automatically begins executing the task.
7. The task completes and its results are mailed to the consumer.
8. A producer leaves the Java Market.

5. Implementation Status & Experimental Results

Current Implementation Status

The Java Market metacomputing system is fully implemented and is publicly accessible at http://www.cnds.jhu.edu/projects/metacomputing. The publicly accessible version differs from the description in this paper only in that it uses simpler decision algorithms within the Market Manager. Optimizing and refining these resource allocation and admission control algorithms is a subject of continuing research within our group. Future releases will incorporate refined versions of the decision making mechanisms described in section 4.3, as well as other potential scheduling policies.

The complete software package is roughly 5000 lines of Java code. This includes about 2000 lines implementing the Task Manager, the Resource Manager and the Market Manager. About 2000 lines implement the Market Library. The additional 1000 lines implement the Request, Launch, and Monitor applets.

Experimental Results

The first major test of the Java Market was performed in the Johns Hopkins Center for Networking and Distributed Systems (CNDS). A CPU-intensive simulation (about 1000 lines of Java code) evaluating five different job scheduling policies for the Mosix system [BGW93] and a stream of jobs was run on the CNDS lab machines. One hundred simulations, each of them representing 10,000 real-time seconds, were run in the following two ways:

1. Running on a standalone Pentium II machine using the Java Developer's Kit.

2. Using the Java Market with six producer machines (two Pentium IIs and four Pentium Pros) using Netscape.

The execution time of the complete simulation on the standalone machine, without compilation and, of course, without remote I/O, was approximately 127 minutes. The execution time of the complete simulation using the Java Market, including downloading, compilation, and remote I/O was approximately 35 minutes. This shows a speedup of about 3.6 on a system with a combined power roughly 4.7 times greater than that of the standalone machine.

Accessing our web site, the interested reader is able to become either a producer or a consumer. Moreover, demo applications, including the above simulation, are available on our web site, ready to be launched.

6. Conclusion

We have built a working system that transfers tasks from any machine to any other machine (with Java-capable browsers) over a network. This structure is easy to use and platform-independent. This system transparently transforms the Internet into a metacomputer with effectively unlimited potential computational power. Further, it can be used within organizations to effectively make use of otherwise wasted CPU cycles. Future work will deal with refined decision making mechanisms, stronger security, and operation in disconnected mode.

References

[AAFL96] B. Awerbuch, Y. Azar, A. Fiat and T. Leighton. Making commitments in the face of uncertainty: How to pick a winner almost every time. In *Proceedings of the ACM Symposium on Theory Of Computing (STOC)*, 1996.

[AAP3] B. Awerbuch, Y. Azar and S. Plotkin. Throughput competitive on-line routing. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, November 1993.

[BGW93] A. Barak, S. Guday and R. Wheeler. The Mosix distributed operating system, load balancing for Unix, Volume 672, May 1993.

[CFGK94] N. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky. Adaptive Parallelism and Piranha. Available at http://www.cs.yale.edu/HTML/YALE/CS/LINDA/ap_and_piranha.html

[CLNR97] N. Camiel, S. London, N. Nisan, O. Regev. The Popcorn Project – An Interim Report, Distributed Computation over the Internet in Java. In the *Proceedings of the 6th International World Wide Web Conference*, April 1997.

[DD97] H. Deitel and P. Deitel. Java: how to program. Published by Prentice Hall.

[FK97] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. In the *International Journal of Supercomputer Applications.* Also available at http://www.globus.org/globus/papers.htm.

[BKKK98] A. Baratloo, M. Karaul, H. Karl, and Z. Kedem. An Infrastructure for Network Computing with Java Applets. In *Proceedings of the ACM Workshop on Java for High Performance Network Computing*. Feb. 1998.

[Java] Sun Microsystems' Java Web site in http://java.sun.com.

[PL95] J. Pruyne and M. Livni. Parallel processing on dynamic resources with Carmi. In *Workshop on Job Scheduling Strategies for Parallel Processing, IPPS*, 1995.