Ph.D. Thesis Proposal

Volunteer Computing

Luis F. G. Sarmenta
(lfgs@lcs.mit.edu)
MIT Laboratory for Computer Science
Cambridge, MA 02139

May 21, 1997

Abstract

The explosive growth of interest in platform-independent programming languages for the World-Wide-Web, such as Sun Microsystems' Java, is opening exciting new possibilites in parallel computing. This thesis will present and investigate the idea of *volunteer computing*, which will allow people to volunteer their computers' processing power towards solving a large parallel problem by simply visiting a web page. Because it requires no prior human contact and very little technical knowledge from the client user, volunteer computing makes it possible to very easily build very large networks of computers working together in parallel. This makes supercomputing more readily accessible and "closer to the masses", and at the same time makes it possible to achieve new heights in performance through the formation of parallel computing networks involving many thousands, or even millions, of computers distributed around the world.

This paper proposes thesis research which aims to expose and investigate the issues and problems involved in implementing and using volunteer computing by developing a working general-purpose volunteer computing programming and execution environment based on Java and using it to create applications of different types. In this paper, we identify some key issues and discuss ways in which they can be handled.

1 Introduction

The introduction of Java by Sun Microsystems has made it possible to place platform-independent executable programs called *applets* on the Web, and let Internet users execute them on their own machines without needing anything more than a Java- capable browser. Applets are now being used by an increasingly large number of people and organizations to enhance their web pages with such things as animation, interactive demos, user- friendly forms, etc. So far, however, most of these uses have only concentrated on providing *local* usefulness—that is, additional functionality or ease-of-use for the client user, but few additional benefits for anyone else. The ability of applets to off-load the server by doing computation on the client's machine, has either been unused, or used only for doing computation relevant to the client alone.

This paper discusses potentials for this unexploited capability of Java, and proposes practical research on a novel form of parallel computing called *volunteer computing*. The idea behind volunteer computing is to allow users from anywhere on the Internet to join in the solving of a big parallel problem by simply using a Java-capable browser and visiting a web site. Because it requires no prior human contact and very little technical knowledge from the client user, volunteer computing makes it possible to very easily build very large parallel computing networks. Potentially, such a network can involve thousands, or even millions, of computers distributed around the world, making it possible to achieve performance levels far beyond that of any current supercomputer. And, since volunteer computing does not require new hardware to be purchased, it can provide affordable supercomputing capabilities even to financially- constrained organizations such as small universities and universities and companies in developing countries. In fact, since volunteer computing makes it easy for organizations to pool together their resources, it opens new possibilities for cooperative and collaborative research between institutions around the world.

This paper proposes a doctoral thesis project which aims to develop a working and usable system that would allow people to develop basic volunteer computing systems, as well as serve as groundwork for further research in the area. In the process of developing such a system, several important issues and problems will be addressed and investigated. These are discussed in the rest of this paper.

Section 2 describes in more detail the motivations for implementing volunteer computing. Section 3 gives an overview of related work already done or currently in progress, and discusses the relationships between the proposed thesis research and these other works. A more detailed description of these works is provided in the Appendix. Section 4 discusses the forseeable problems and issues that need to be addressed, and proposes possible directions for research work. Section 5 presents a tentative schedule for addressing these issues and developing a working volunteer computing system. Finally, we conclude with a short summary in Section 6.

2 Motivations

2.1 Current Systems

Volunteer computing is actually just a new variation on an old idea: that of using a network of workstations, or NOW, to solve a parallel problem. The idea of NOWs first became popular because it allowed people to take advantage of their existing (and mostly- idle) workstations, enabling them to do parallel processing without having to purchase an expensive supercomputer. Global- scale NOWs, employing computers geographically distributed around the world and communicating through the Internet, have been used with great success to solve large parallel problems as far back as the early 90's [?, ?, ?], and until today [?, ?].

Unfortunately, many, if not most, of these projects have used *ad hoc* software systems. Typically, a subsystem for providing communication and coordination between machines in the NOW had to be developed mostly from scratch as part of each project. Furthermore, in many cases, the systems are not even fully automatic. Participants have to *manually* request jobs, load them into their computers, execute them, and again manually submit the results [?]. Thus, while these software systems can and have been used for NOWs containing several thousands of workstations, doing so requires a large amount of human effort in terms of setting up, coordinating, and administering the system.

In recent years, this situation has been improved by the development of *general-purpose* parallel processing systems such as PVM [1] and MPI [?].[1] In such systems, the amount of manual work required is reduced significantly by the runtime system, which takes care of such things as automatically executing the appropriate code on each of the processors involved, keeping track of existing processors, routing and delivering messages between processors, performing broadcasts and barrier synchronization, etc. At the same time, programming is made much easier by a general-purpose applications programming interface (API) which hides most of the detail of the runtime system, and allows the user to write parallel programs for NOWs using a relatively simple high-level message-passing model.[2] All this allows programmers to not worry about the low-level details of communication and coordination between the processors, and instead concentrate on writing applications.

2.2 Problems with Current Systems

Although systems like PVM and MPI make programming and setting-up NOWs significantly easier that it was with the earlier ad hoc systems, setup requirements still impose practical limits on the size of NOWs that can be used with these systems. In order to perform a parallel computation using the popular Parallel Virtual Machine (PVM) library [1], for example, one needs to take the following steps:

1. Install the PVM daemon on all machines to be used in the computation.
2. Compile binaries for each target architecture.
3. Distribute these binaries to all the machines (either by explicitly copying them, or by using a shared filesystem).
4. Provide the owner of the computation with an account and password on all the machines.
5. Provide the owner of the computation with remote shell access (i.e., the ability to execute shell functions and programs remotely) on all machines. This is used to execute the PVM daemon and the application binaries on the remote machine.

It is not hard to see that this process not only requires a lot of effort on the system administrator's part, but also requires a lot of trust between the involved machines, and consequently, a lot of prior human communication between the administrators of different machines.

In contrast, a Java-based implementation of volunteer computing would require very little effort to set-up. The steps above are addressed as follows (for a summary of Java's capabilities see section **??**):

1. Install a Java-capable browser on all machines. Since most popular browsers today support Java, this is practically a given on any machine with World-Wide-Web access.
2. Compile binaries (only once!) into machine-independent Java applet code.

[1] For some examples of applications using PVM and MPI see [?, ?, ?].

[2] In fact, the PVM and MPI APIs were designed to facilitate programming for *all* kinds of message-passing parallel hardware, not just NOWs.

3. Place the Java applet on a web server, and then advertise the web page to potential volunteers. Volunteers can automatically download and execute the applet code by simply browsing the web page containing the applet.

4. The owner of the computation does *not* need an account on any of the machines, except maybe the server itself.

5. There is no need to provide remote shell access to any of the machine, either. The applet is already automatically executed by the browser. Furthermore, unlike with remote shell access, the owner of the computation does not get permission to access or execute anything on the remote machines other than the Java applet itself.

Thus, unlike PVM and similar systems, a Java-based volunteer computing system requires minimal setup and distribution effort, minimal technical knowledge on the client side, and minimal security risk.

[The following is taken verbatim from an old version of this paper and should be rewritten or deleted.]

The characteristic ease with which a volunteer computing system can be set-up makes it possible to form very large computing networks without any prior human contact between the owner of the computation and the owners of the volunteer workstations. (In fact, an *ideal* volunteer computing system would still work even if the server and the client are allowed to mutually distrust each other in the computer security sense.) With volunteer computing, the setup time for a system is basically limited only by the time it takes to invite people to access the home page.

From the experience of popular web sites, we know that given enough public interest, it is not impossible to get thousands, maybe even millions, of clients in a very short time. During the Kasparov-Deep Blue match, for example, the IBM web server dedicated to providing live updates on the match received 5 million hits on the first day [2].[3] Since a user usually generates many hits when visiting a web page, this figure does not mean 5 million users. However, even generously allowing for 1,000 hits per user, this still results in 5,000 users in one day— a number greater (as far as I know) than the size of the largest NOWs formed to solve a parallel problem to-date.[4]

This ability to very easily construct such truly ”massive” world- wide networks-of-workstations opens up new possibilities in parallel computing, making it possible to consider solving problems too large to be considered before. As volunteer computing makes it very easy for people around the world to traverse political and geographic boundaries, and cooperate on achieving a common goal, it also potentially has significant positive social consequences.

[3]The IBM server actually went down on the first day due to the unmanageable demand on it (the site was originally expecting only 200,000 ”visits” per day). It was quickly redesigned to handle 400,000 hits per hour, and was successfully used for the rest of the match [2].

[4]The network that factored RSA-129 involved over 1,600 computers [3].

3 Related Work

[This section is taken verbatim from a draft written in October, 1996, and needs to be updated. Notably missing here are the recent results of the work being done by Javelin group at UCSB and the Charlotte group at NYU.[5]]

3.1 Very Recent Work

In early September 1996, a number of papers were presented at the ACM SIGOPS European Workshop [18] that are likely to be of relevance to volunteer computing research. The ATLAS system by Baldeschwieler, Blumofe, and Brewer [19] uses techniques developed for Cilk [20] with Java to create a Java-based NOW system with efficient thread- scheduling. The ParaWeb project by Brecht, et al. [21] presents another Java-based NOW system employing a parallel class library that allows threads to be executed in parallel on several Java virtual machines. Both ATLAS and ParaWeb work under the premise that there are dedicated, high-performance, *compute servers* available on the Net, and that clients use the system (either ATLAS or ParaWeb) by submitting work in the form of threads to a scheduling server, which takes care of scheduling threads on the compute servers. It is not clear to me right now whether these systems allow the clients to act as computer servers themselves. (I don't think so.) Also, as far as I know, both implementations, use Java *applications*, for the computer serve, not applets. Another project, Legion, by Grimshaw et al. [22], does not use Java, but is nevertheless intriguing because it sets as its goal the "realization of a worldwide virtual computer". Legion has been around for over a year now, and is being tested in a number of government labs. Klaus Schauser, from UCSB, gave a talk on "Global Computing: A Research Agenda for the Next Millenium" in June 1996 at UCSB about a research project whose goals are very similar to those of volunteer computing mentioned in this paper. However, I have no information on this project other than the abstract of his talk [23].

A number of other papers involving parallel and distributed computing using the Web and/or Java were presented in HPDC-5 in early August 1996 [24]. However, I have not gotten a copy of the proceedings yet.

3.2 Developments in Industry

[Talk about RMI, Java Beans, JIT in Netscape 3.0 and Microsoft IE 3.0, ActiveX, JDBC, etc.]

3.3 JavaPVM and JPVM

Two interesting PVM-related packages for Java have recently been made available on the Web: JavaPVM, and JPVM. JavaPVM, by David Thurman [25], is an interface which allows Java programs to call PVM functions as native methods. This allows one to use Java to write programs that can interact with the PVM daemon, and therefore with other PVM programs as

[5] See http://www.cs.ucsb.edu/research/superweb/ and http://cs.nyu.edu/milan/charlotte/

well. Note, however, that the PVM daemon is *not* written in Java, and that the PVM methods are *not* portable. JPVM, by Adam Ferrari [26], is a library which gives PVM- like functionality to Java applications. It is "PVM-like" because it is *not* interoperable with PVM applications. However, it is completely written in Java, so it is portable, unlike JavaPVM. I have not yet tried to use JPVM, but if it can be adapted for use with applets (and not just applications), then it may become very useful as a preliminary working API for prototype volunteer computing programs.

3.4 WebWork

WebWork is a joint project of Syracuse University, Boston University, and Cooperating Systems Corporation. It seeks to develop an "extensible world-wide virtual machine (WWVM) model for heterogeneous and distributed high performance computation," which it intends to use for solving Grand and National Challenges. They already have a working RSA factoring demonstration, and are targetting other problems such as Telemedicine, and Chemistry. As of their June 14, 1995 report, [28], which is the latest I could find so far, they have not yet used Java in the ways I have proposed in this paper. Their system works by using CGI, HTTP, HTML, and MIME to allow web *servers* to exchange data. As far as I can tell, they have not implemented anything where computation is done by *clients*. In their report, which was written only shortly after the initial release of Java, they have expressed interest in using Java. However, it seems that they only intend to use it for improving the client's user interface, rather than for doing computation on client machines.

3.5 RSA Factoring

A crude form of volunteer computing (which does not use Java, and still requires human coordination and trust) is currently being used to do RSA Factoring on the Web [29, 30]. The idea of using Java applets for factoring is not a unique one. DigiCrime, a web site which presents Internet security concerns in a lighthearted manner, alludes to the possiblity of "stealing" people's processing power by embedding Java applets which do computation such as factoring, into popular websites [31]. Two students at the University of Washington have actually implemented such a system, and have made their source code available for downloading [32].

3.6 Work by PCRC

The Parallel Compiler Runtime Consortium (PCRC) has come-up with a preliminary report on "HPCC and Java" [33]. This report presents a number of interesting preliminary ideas about using Java for distributed high-perfomance computing for use with Grand Challenges, National Challenges, and "Meta-Challenges".[6] The way they plan to use Java is by extending it into a distributed object language. They discuss several possible directions, including ways to make it compatible with CORBA, the industry standard for distributed object management. They also propose some ideas on adding parallel processing support to the Java language, compiler, and

[6]These are "loosely coupled sets of physically distributed instances of Grand or National Challenges," such as distributed interactive simulation problems

interpreter. One interesting idea presented in their report is that of *itinerant programs*, which move from server to server depending on the availability of data and processing power. This seems to be very similar to the pool-of-volunteers model discussed above. However, it is not clear from their report whether they have thought of using true volunteers as servers in their proposed system, and of the security issues involved.

3.7 Other Work on Distributed Java

Sun Microsystems has been developing NEO and Joe, which seem to provide distributed object capabilities to Java [34]. JIDL from Sandia, and HORB from ETL in Japan, are two separate projects that seek to link Java and CORBA [35, 36]. A group in Australia is experimenting with implementing MPI on Java [37].

3.8 Relation to Proposed Volunteer Computing Project

It is clear that this volunteer computing project would not be the first attempt to use Java for parallel and distributed computing on the World-Wide-Web. However, I think that this field is still relatively new, and there is still a lot of space to do original and pioneering work. In particular, I imagine this project's place to be somewhere between the early prototypes such as WebWork and University of Washington's Java RSA factoring project, and the more general and ambitious goal of using Java for distributed object computing. This project will improve on the efforts done by the early prototypes (possibly leveraging from their results), in order to generate a parallel processing system that is bigger in terms of nodes it can handle, and more flexible in terms of parallel algorithms it can support. It does not aim to produce a system flexible enough to handle the truly distributed object- oriented systems that PCRC and others are targetting, but at the same time, it can produce results which would be of interest to these research groups.

4 Proposed Work

4.1 Objective

The objective of this thesis is to expose and investigate the issues and problems involved in volunteer computing by developing a usable volunteer computing programming system and using it to create applications of various types. Some of the forseeable issues and problems to be addressed are discussed in the following sections.

4.2 Adaptive Parallelism

- Computation must not depend on the knowledge of the number of nodes, the architecture of the nodes, or the structure of the interconnection network.
  - Programming interface.
  - Performance must be reasonable despite nodes dropping in and out.
- Possible Approaches
  - Cilk (ATLAS)
  - Linda / Dataflow (Javelin, WWWinda, Jada)
  - Eager Scheduling (Charlotte, my first demo)
  - Others?

4.3 Fault-Tolerance

- stopping faults – fall under adaptive parallelism.
- Byzantine faults – the harder problem
  - malicious
    * intentional sabotage
    * cheating by laziness (in paid volunteer systems)
    * spoofing
  - not malicious
    * faulty processors (e.g. Pentium)
    * faulty network links (losses, corruption, etc.)
- Possible Approaches
  - Checksums and Digital Signatures
  - Encrypted Computation
    * worker does not know what it is computing so cannot alter computation
    * also useful for preventing spying in market systems

- Redundancy and Randomized Algorithms
    - TMR-type redundancy
    - Spot-checking – for each problem, there is probability P that the work will be rechecked. If you are found to be wrong, then you're in big trouble.
- Blacklisting – nodes or domains identified as faulty a certain number of times will be ignored and not allowed to join in computation.
- Problem Choice
    - Search problems with easily-verifiable results.
    - Problems that do not require 100% accuracy (e.g., graphics, video, sound, etc.)

4.4 Applications

- Finding Good Application Domains
    - Some are better suited to volunteer computing that others
    - Probably good
        - Search problems (cryptography, number theory, etc.)
        - Graphics (rendering, Mandelbrot-like graphics, etc.)
    - Probably bad
        - Scientific computations which are fine-grain and need high accuracy
    - Other interesting possibilities
        - Computer Chess (i.e. Kasparov vs. The Whole World)
        - Distributed Simulations (e.g., traffic) – nice to do with remote objects!
    - Attracting and motivating people to join computations is also an issue.
- Economic Model (Applications for the Real World)
    - True (altruistic) volunteers
    - Forced volunteers
        - all volunteers are "owned" by same group
        - useful within universities or companies
        - network-of-information-appliances
    - Paid volunteers
        - individual compensation (money/credits, discounts, lottery, promos, etc.)
        - market system (cite Javelin)

4.5 Other (Secondary?) Issues

- Programming Interface
- User-Interface
- Congestion and Scalability
- Hidden Costs

5 Schedule

6 Conclusion

References

[1] Al Geist et al. *PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallelism.* MIT Press, 1994.
URL: `http://www.netlib.org/pvm3/book/pvm-book.html`

[2] IBM Corporation. IBM Intranet and Client/Server Solution—Deep Blue Versus Kasparov on the Internet. HTML document, Sept. 1996.
URL: `http://www.csc.ibm.com/advisor/provensolutions/pcid/380e_362.html`

[3] Steven Levy. Wisecrackers. *Wired*, Issue 4.03, March 1996.
URL: `http://www.hotwired.com/wired/4.03/features/crackers.html`

[4] Ken Arnold and James Gosling. *The Java Programming Language.* Addison-Wesley Publishing Company, Inc., 1996.
URL: `http://www.aw.com/cp/arnold-gosling.html`

[5] Sun Microsystems, Inc. Remote Method Invocation. HTML document, Sept. 1996.
URL: `http://chatsubo.javasoft.com/current/rmi/index.html`

[6] Sun Microsystems, Inc. JDK 1.1 Preview. HTML document, Sept. 1996.
URL: `http://java.sun.com/products/JDK/1.1/designspecs/index.html`

[7] Ann Wollrath, Sun Microsystems, Inc., Personal communication, Java Day at MIT, Sept. 20, 1996.

[8] University of Illinois at Urbana-Champaign. Grand and National Challenges. HTML document, Oct. 1995.
URL: `http://www.ncsa.uiuc.edu/Cyberia/MetaComp/GrandNat.html`

[9] RSA Data Security. RSA Factoring Challenge. HTML document.
URL: `http://www.rsa.com/factor/chalenge.htm`

[10] Charles Perkins. *Java Unleashed*, chapter 39. Sams.net Publishing, 1996.

[11] Pendragon Software. The Java Performance Report. HTML document, July 1996.
URL: `http://www.webfayre.com/pendragon/jpr/index.html`

[12] Mark LaDue. Hostile Applets Home Page. HTML document, June 1996.
URL: `http://www.math.gatech.edu/~mladue/HostileApplets.html`,

[13] Drew Dean, Edward W. Felten, and Dan S. Wallach. Java Security: From HotJava to Netscape and Beyond, in *Proceedings of the IEEE Symposium on Security and Privacy*, May 1996.
URL: `http://www.cs.princeton.edu/sip/Publications.html`

[14] Douglas Kramer. The Java Platform: A White Paper. Sun Microsystems, Inc. HTML Document, Aug. 1996.
URL: `http://java.sun.com/doc/white_papers.html`

[15] Bill Gates. *The Road Ahead.* Viking, a division of Penguin Books USA Inc., 1995.

[16] Nicholas Negroponte. *Being Digital.* Vintage Books, a division of Random House, Inc., 1995.

[17] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kauffman Publishers, Inc., 1996.

[18] *Proceedings of the Seventh ACM SIGOPS European Workshop: Systems Support for Worldwide Applications.* ACM Special Interest Group on Operating Systems. Sept. 9-11, 1996, Connemara Ireland.

[19] J. Eric Baldeschwieler, Robert D. Blumofe, and Eric A. Brewer. ATLAS: An Infrastructure for Global Computing, in *Proceedings of the Seventh ACM SIGOPS European Workshop: Systems Support for Worldwide Applications*, Sept. 1996.

[20] Blumofe, et al. Cilk: An Efficient Multithreaded Runtime System, in *Proceedings of the 5th ACM SIGPLAN Symposium on Principles of Parallel Programming* (PPOPP '95), July 19-21,1995, Santa Barbara California, pp. 207-216.
URL: `http://theory.lcs.mit.edu/~cilk/`

[21] Tim Brecht, et al. ParaWeb: Towards World-Wide Supercomputing, in *Proceedings of the Seventh ACM SIGOPS European Workshop: Systems Support for Worldwide Applications*, Sept. 1996.

[22] Andrew S. Grimshaw and William A. Wulf. Legion: Flexible Support for Wide-Area Computing, in in *Proceedings of the Seventh ACM SIGOPS European Workshop: Systems Support for Worldwide Applications*, Sept. 1996.
URL: `http://www.cs.virginia.edu/~legion/`

[23] Klaus E. Schauser. Global Computing: A Research Agenda for the Next Millenium. Talk given at U.C. Santa Barbara, on June 3, 1996.
URL: `http://www.cs.ucsb.edu/colloquia/Abstracts/Schauser.html`

[24] *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, Aug. 9-11, 1996, Syracuse, New York.
URL: `http://www.npac.syr.edu/projects/hpdc`

[25] David A. Thurman. JavaPVM: The Java to PVM Interface. HTML Document, June 1996.
URL: `http://homer.isye.gatech.edu/chmsr/JavaPVM/`

[26] Adam Ferrari. JPVM: The Java Parallel Virtual Machine. HTML document, June 1996.
URL: `http://www.cs.virginia.edu/~ajf2j/jpvm.html`

[27] Northeast Parallel Architectures Center. Web-based HPCC at NPAC. HTML document, July 1996.
URL: `http://www.npac.syr.edu/projects/webbasedhpcc/index.html`

[28] Geoffrey Fox et al. WebWork: Integrated Programming Environment Tools for National and Grand Challenges. Technical Report SCCS-715, Northeast Parallel Architectures Center, June 1995.
URL: `http://www.npac.syr.edu/projects/webbasedhpcc/index.html`

[29] Cooperating Systems Corporation. FAFNER: Factoring via Network-Enabled Recursion. Perl-generated HTML document, 1996.
URL: `http://www.cooperate.com/cgi-bin/FAFNER/factor.pl`

[30] Northeast Parallel Architectures Center. RSA Factoring-By-Web Project. HTML Document, Jan. 1996.
URL: http://www.npac.syr.edu/factoring.html

[31] DigiCrime Computational Services via Java. HTML Document, June, 1996.
URL: http://www.digicrime.com/java.html

[32] Geoff Voelker and Dylan McNamee. The Java Factoring Project. HTML document with Java Alpha version applets, Sept. 1995.
URL: http://www.cs.washington.edu/homes/dylan/ContestEntry.html

[33] Parallel Compiler Runtime Consortium. HPCC and Java—A Preliminary Report. HTML Document, May, 1996.
URL: http://www.npac.syr.edu/users/gcf/hpjava.html

[34] Sun Microsystems, Inc. Sunsoft NEO Product Family. HTML document, Aug. 1996.
URL: http://www.sun.com/sunsoft/neo/

[35] Sandia National Labs. JIDL: The CORBA IDL Compiler for Java. HTML Document, June 1996.
URL: http://herzberg.ca.sandia.gov/jidl/

[36] Hirano Satoshi. HORB home page. HTML Document, Sept. 1996.
URL: http://ring.etl.go.jp/openlab/horb/

[37] Sam Taylor. Prototype Java-MPI Package. HTML Document, April 1996.
URL: http://cisr.anu.edu.au/~sam/java/java_mpi_prototype.html