2008-03-15

Improving the Productivity of Volunteer Computing

David M. Toth
Worcester Polytechnic Institute

Improving the Productivity of Volunteer Computing

by

David M. Toth

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Computer Science

March 16, 2008

APPROVED:

____
Professor David Finkel, Advisor

____
Professor Mark Claypool, Committee Member

____
Professor Craig Wills, Committee Member

____
Professor Xiannong Meng, External Committee Member
Bucknell University

____
Professor Michael Gennert, Department Chair

Abstract

The price of computers has dropped drastically over the past years enabling many households to have at least one computer. At the same time, the performance of computers has skyrocketed, far surpassing what a typical user needs, and most of the computational power of personal computers is wasted. Volunteer computing projects attempt to use this wasted computational power in order to solve problems that would otherwise be computationally infeasible. Some of these problems include medical applications like searching for cures for AIDS and cancer. However, the number of volunteer computing projects is increasing rapidly, requiring improvements in the field of volunteer computing to enable the increasing number of volunteer projects to continue making significant progress.

This dissertation examines two ways to increase the productivity of volunteer computing: using the volunteered CPU cycles more effectively and exploring ways to increase the amount of CPU cycles that are donated. Each of the existing volunteer computing projects uses one of two task retrieval policies to enable the volunteered computers participating in projects to retrieve work. This dissertation compares the amount of work completed by the volunteered computers participating in projects based on which of the two task retrieval techniques the project employs. Additional task retrieval policies are also proposed and evaluated. The most commonly used task retrieval policy is shown to be less effective than both the less frequently used policy and a proposed policy. The potential that video game consoles have to be used for volunteer computing is explored, as well as the potential benefits of constructing different types of

volunteer computing clients, rather than the most popular client implementation: the screensaver.

In addition to examining methods of increasing the productivity of volunteer computing, 140 traces of computer usage detailing when computers are available to participate in volunteer computing is collected and made publicly available. Volunteer computing project-specific information that can be used in researching how to improve volunteer computing is collected and combined into the first summary of which we are aware.

Acknowledgements

I would like to thank everybody who has made this dissertation possible. The biggest thanks go to my wife, Laura, whose love, understanding, and sacrifices made it possible for me to go back to school full time for 5 years. Laura put up with all sorts of things during my time in school, from the long hours, late nights, and occasional all-nighter, to me turning from a neat freak into a slob during school. She has also endured a non-stop stream of babble about my dissertation and she probably knows more about my work than everyone other than my advisor does.

I would also like to thank my advisor, Professor David Finkel. His willingness to work with me on this project allowed me to work on a topic that truly fascinated me, as opposed to other students who have to work on topics that are of more interest to their advisors than to themselves. Professor Finkel has guided me along the path to the Ph.D., providing invaluable insight and advice all along the way. He has helped me to understand the nature of research much better. I truly could not have asked for a better advisor.

I would like to thank my dissertation committee, Professors Mark Claypool and Craig Wills of WPI and Professor Xiannong Meng of Bucknell University. My committee members provided a lot of insight and suggestions that helped improve the quality of this work.

Thanks to my family, both close and extended. Your support and understanding through these years has made it much easier to focus on school and finish in a timely manner. Special thanks go to ***Katie***, my loyal greyhound, for helping me through an extremely challenging semester in the fall of 2006 when I needed it.

I would like to thank all my friends, but most especially Mark Giles, David Shapiro, and Seth Webster. We managed to find time every few weeks for gaming, and of course to discuss my work over dinner beforehand. ☺ Thanks to my friends who were fellow students at WPI (even if you all did graduate with your Masters Degrees and leave me there by myself to finish my Ph.D. alone), Kerry McKay, Frank Posluszny III, and Adam Zellers.

I would like to thank the professors from Connecticut College and WPI who touched my life. Bridget Baird, Ozgur Izmirli, Kathy McKeon, Gary Parker, and Perry Susskind inspired me as an undergraduate and helped me get into graduate school. Dan Dougherty, Mike Gennert, Rob Lindeman, Gary Pollice, and Stan Selkow were wonderful friends and advisors throughout my time at WPI. I cannot thank you enough for all your help and support.

Many people helped with the technical aspects of this work, providing knowledge, equipment, technical support, and of course data sources. Jesse Banning, Mike Voorhis, and Josh Brandt provided lots of equipment so I could collect the data I needed, run my simulations, and work on other portions of the dissertation. Jesse, Mike, and Chuck Wilcox provided lots of technical support, too. Dave Anderson, a student at WPI provided help with the Windows API. I would like to thank all of the people who allowed me to collect data from their computers. Without that data, I could not have completed this work. A special thank you is in order to Tom Collins, who enabled me to collect the data I needed from the public workstations at WPI. I cannot thank the friend and one of his co-workers (at a company that wished to remain anonymous) enough for allowing me to collect data about the usage patterns of some of the company's computers. You two know who you are and I am sorry I cannot acknowledge you publicly. Finally, I would like to thank Dr. David Anderson from UC Berkeley, Dr. Derrick Kondo, and George Woltman of the Great Internet Mersenne Prime Search (GIMPS) for answering all of my questions.

I would like to thank some of the WPI staff members who contributed to this dissertation indirectly and made my time at WPI more pleasant. Christine Drew, who works in the library at WPI, took a bigger interest in my work than anybody else not already mentioned, which was encouraging. There are so many projects at WPI and it seems like very few people take an interest in work outside of there area that is not explained with pretty pictures or associated with large grants. For somebody outside my department to express as much interest as Christine did made me feel like I was working on something that matters. I would also like to thank Kim Richardson and Jeannette Dailida in the registrar's office. These wonderful women always cheerfully helped me get my registration straightened out and provided transcripts and everything else I needed during my five years, allowing me to focus on my work, rather than the mounds of forms which I never managed to figure out during my five years at WPI. I could always count on a friendly greeting when I went to the registrar's office and always left knowing that everything was all set.

A project this big that spans several years does not succeed without the help of many people, and it seems that inevitably, somebody's help goes unacknowledged. I would like to apologize to anybody who I left out of these acknowledgements. I assure you that it was unintentional and I valued your help greatly.

Table of Contents

1 Introduction

1.1 Volunteer Computing

Volunteer computing is a type of distributed computing that uses volunteered computational resources in order to accomplish some goal [1]. Large volunteer computing projects began in the 1990's with The Great Internet Mersenne Prime Search (GIMPS) and Distributed.net [1]. These projects allow people to solve many problems that were previously computationally infeasible. The power of volunteer computing is the ability to perform millions of tasks simultaneously. Problems that can be decomposed into many independent tasks can take advantage of millions of computers working simultaneously. More than 20 volunteer computing projects are currently running, including the well-known SETI@home project, which searches for evidence of extraterrestrial life [2].

A volunteer computing project uses a set of servers to create, distribute, record, and aggregate the results of a set of tasks that the project needs to perform to accomplish its goal. The tasks are evaluating data sets, called *workunits* [3]. The servers distribute the tasks and corresponding workunits to clients (software that runs on computers that people permit to participate in the project). When a computer running a client would otherwise be idle (in the context of volunteer computing, a computer is deemed to be idle if the computer's screensaver is running), it spends the time working on the tasks that a server assigns to the client. When the client has finished a task, it returns the result obtained by completing the task to the server. If the user of a computer that is running a client begins to use the computer again, the client is interrupted and the task it is processing is paused while the computer executes programs for the user. When the computer becomes idle again, the client continues processing the task it was working on when the client was interrupted.

Volunteer computing has enabled researchers to solve problems that were previously computationally infeasible by decomposing them into smaller problems, distributing the smaller problems to volunteer computers, and aggregating the results returned by the computers to form the solution to the large problem. The applications of volunteer computing include solving problems in the medical, scientific, and mathematical fields. Currently, volunteer computing programs are searching for cures for diseases, looking for evidence of extraterrestrial intelligence, and finding Mersenne prime numbers, in addition to many other applications [1, 2, 4]. Several other projects have been completed by volunteer computing programs, including several encryption challenges that were solved by distributed.net [5, 6, 7].

Between 1996 and 2000, there was a flurry of activity on research including and related to volunteer computing, including ATLAS [42], Charlotte [8], ParaWeb [9], Javelin [10], Popcorn [11], Bayanihan [12], Distriblets [13], Gucha [14], and others [12]. By 2000, interest in the field began to wane and shifted towards the more lucrative area of grid computing, and only a few research projects remained active. Both the BOINC [15] and Xtremweb [16, 17, 18] projects provide a framework, or a set of tools to assist in the creation of volunteer computing projects. BOINC and Xtremweb continue to move forward, with BOINC having become the dominant framework to build volunteer computing projects. Nevertheless, even with the significantly smaller number of research projects in the area, many volunteer computing projects are still running, with new ones springing up.

In recent years, the prices of personal computers have dropped drastically. In fact, it is routine to see a computer on sale at stores like Best Buy and Circuit City for $300 in their weekly circulars. The price drops have made it possible for many households to have one or more computers. In addition to family computers, many college students have their own computers and the large number of colleges throughout the United States have many computers in labs and libraries for student use. A lot of these computers are connected to the Internet and many computers remain turned on 24 hours a day, only being used for part of the day and sitting idle during the rest of the day. These computers have the potential to be used for volunteer computing during those idle periods. Additionally, there is a trend of putting more than one core on a CPU so that personal computers are now coming with multiple processing cores instead of a single one. While personal computers with these multi-core CPUs are now truly capable of doing multiple tasks at the same time, people still have trouble doing multiple tasks at once. Therefore, we expect that there will be an increase in the number of CPU cycles that are used by the system's idle process. Other people than the computers' owners could potentially use these CPU cycles without noticeably affecting the computers user's experience. These changes have the potential to really increase the amount of CPU cycles donated to projects by volunteer computing clients.

1.1.1 Summary of Volunteer Computing Projects

There are a number of volunteer computing projects currently running and several that have been completed. This section attempts to give a flavor of the different types of projects and discusses some of the most influential ones.

1.1.1.1 The Great Internet Mersenne Prime Search

George Woltman started the Great Internet Mersenne Prime Search (GIMPS), the oldest of the major volunteer computing projects, in 1996 [1, 19]. In the ten years since this volunteer computing project has been running, the computers participating in it have discovered ten previously unknown Mersenne primes, the most recently discovered one being the largest known prime number [4].

1.1.1.2 Distributed.net

Distributed.net, the organization that started the second large volunteer computing project, began in 1997 [1]. Since then, they have run nine volunteer computing projects split between two categories: optimal mark golomb rulers and cryptography [20]. Optimal mark golomb rulers are used for "sensor placements for X-ray crystallography and radio astronomy. Golomb rulers can also play a significant role in combinatorics, coding theory and communications" [21]. Distributed.net used their volunteer computing projects to win several cryptographic challenges, including the CS-Cipher Challenge, and four challenges sponsored by RSA Laboratories [20].

1.1.1.3 SETI@Home

SETI@home [22] is probably the most well known volunteer computing project. The project, conceived in 1995 by David Gedye and started in May 1999, searches for extraterrestrial life by processing the signals collected by radio telescopes [23]. In addition to just scanning the data for signs of extraterrestrial life, the Seti@home client program displays a screen saver showing information about the signals it is currently processing [1]. As of July 4, 2005, over two million years of CPU time had been contributed to Seti@home [24].

1.1.1.4 Grid.org

Grid.org is an organization that provides a web site that acts as a centralized place for several large medical volunteer computing projects and a location where people can go to have their computers participate in the projects [25]. The first project Grid.org worked on was a cancer research project that Intel sponsored [26]. In the project, "grid.org was able to screen billions of target molecules against known cancer target proteins" [26]. Following the cancer project, grid.org worked on projects to try to identify treatments for smallpox and anthrax [26]. Currently, Grid.org is working on analyzing human proteins [26].

1.2 Dissertation Motivation

There are several issues that have motivated this dissertation: 1) the potential benefits to society that can be realized by people participating in volunteer computing projects, 2) the small number of participants in volunteer computing projects, and 3) the increasing number of projects as people have discovered the viability of volunteer computing projects. Clearly, finding cures to diseases such as AIDS and cancer could improve the quality of life for millions of people, and many other mathematical and scientific advances could also help the world. Therefore, it is obvious that increasing the effectiveness of volunteer computing projects is a worthwhile goal, as speeding up the discovery of these advances will allow the discoveries to benefit more people.

The second motivating factor for advancing increasing the effectiveness of volunteer computing projects is that while many idle computer cycles have been harnessed for volunteer computing projects, statistics show that very few people participate in volunteer computing projects. In April, 2005, there were fewer than 8 million users for the volunteer computing projects of which we are aware, which includes users who no longer participate and likely users who have been counted in multiple projects [27, 28, 29, 30, 31, 32, 33, 34, 35, 24]. In contrast to this, the number of people who have participated in volunteer computing projects, the estimated number of hosts connected to the Internet exceeded 317 million in January 2005 [36]. Fewer than 1% of an estimated 300 million Internet-connected personal computers participate in volunteer computing projects [37]. Increasing the effectiveness of the CPU cycles that are donated will help offset the low participation rate.

The growing number of volunteer computing projects has significantly raised the amount of CPU time that must be donated to make progress on all of the existing projects. In addition to the increasing number of projects, while some projects end at the completion of a task that is known to be solvable in a finite amount of time, other projects are designed to continue indefinitely, continually searching for more or better information. Therefore, not only has the number of projects continued to grow, but because many of them may never be completed, the demand for donated CPU cycles will never end. Due to the benefits of some projects, the low participation rate, and the infinite nature of some projects, it is crucial to get more people to donate CPU cycles and that volunteer computing projects utilize the donated CPU time as efficiently as possible in order to maximize the amount of information they collect.

1.3 The Dissertation

This work aims to enable those building volunteer computing projects and frameworks that assist in the creation of volunteer computing projects to increase the effectiveness of volunteer computing projects. In order to increase the effectiveness of volunteer computing, we examine two ways to increase the productivity of volunteer computing: using the volunteered CPU cycles more effectively and exploring ways to increase the amount of CPU cycles that are donated.

1.3.1 Using Volunteered CPU Cycles More Effectively

In an attempt to identify a way to improve volunteer computing projects so they could use donated cycles more effectively, we looked for an aspect of volunteer computing that was not standardized. We found that each of the existing volunteer computing projects uses one of two task retrieval policies to enable the volunteered computers participating in projects to retrieve work. The first task retrieval policy instructs each volunteered to retrieve one tasks at a time from the volunteer computing project's server. When a volunteered computer completes the task and returns the result to the volunteer computing project's server, the volunteered computer retrieves another task. We refer to this policy as *Buffer None*. The second task retrieval policy instructs each volunteered computer to retrieve multiple tasks from the volunteer computing project's server at a time. The tasks are stored in a buffer and processed one at a time. When a task is completed, the result is sent to a server and the next task in the buffer is started. When the buffer is almost emptied, the volunteered computer retrieves another set of tasks from the server. We refer to this policy as *Buffer N Days* because the volunteered computer buffers some number, *N*, days of work. Because two different task retrieval policies were in use and we were unable to find information about the advantages and disadvantages of each policy, we suspected that the task retrieval policy was an aspect of volunteer computing that had not been studied thoroughly. After examining how each policy worked, we believed that using one of the two different task retrieval policies might result in more donated CPU cycles being wasted than using the other policy. We also suspected that by combining the best features of both policies, we might be able to produce a new policy that could result in even fewer donated CPU cycles being wasted.

1.3.2 Exploring Ways to Increase Donated CPU Cycles

In an attempt to increase the number of CPU cycles that are donated to volunteer computing projects, we explored the potential of porting volunteer computing clients to devices besides general-purpose computers. Specifically, we tested how the productivity a client running on a video game console would compare to a client running on a general-purpose computer. We also attempted to determine what the performance difference would be if volunteer computing clients ran constantly on a computer as a service/daemon process in the background, as opposed to running as a screensaver which is the most common implementation.

1.4 Contributions

The contributions of this dissertation are:

1. We have used simulation to compare the task retrieval methods used in different volunteer computing projects to find which ones result in the highest productivity of volunteer computing projects.
2. We have proposed alternate task retrieval policies and evaluated them. We found that one of our proposed task retrieval method outperforms the other methods in our simulations.
3. We have explored the potential benefit of using video game consoles to run volunteer computing clients.
4. We have explored the increase in productivity of volunteer computing projects if volunteer computing clients ran constantly in the background as a service/daemon process instead of running as a screensaver.
5. We have collected 140 traces of computer usage activity for 28 days from home, business, public access, and undergraduate student computers. We have made these traces publicly available for others to use. The traces are contained in the traces.zip file that can be downloaded from the same place you downloaded this dissertation.

6. We have collected publicly available information about real values used for volunteer computing projects and combined it into the first summary of volunteer computer project values of which we are aware.

1.5 Organization of Dissertation

The remainder of this work begins by presenting background about distributed computing and details about how volunteer computing works in Chapter 2. We present different types of distributed computing, the anatomy of volunteer computing projects, some of the existing volunteer computing project frameworks, and discuss the design decisions that people must make when creating volunteer computing projects. In Chapter 3, we discuss the social and technical ways to decrease the time it takes for volunteer computing projects to complete and the tradeoffs involved in volunteer computing projects. Chapter 4 explains the data we needed in order to analyze the effects of projects using different task retrieval policies and different types of clients and how we obtained the data. Chapter 5 discusses the simulator we created to evaluate how much work volunteer computing clients would complete using different task retrieval policies. Chapter 6 presents our results of using different task retrieval policies. Chapter 7 explains how we validated our simulations. Chapter 8 discusses the effects of using different types of volunteer computing clients. Chapter 9 presents our conclusions and in Chapter 10, we discuss areas we believe are worthy of further study.

2 Background

2.1 Distributed Computing

In the past, personal computers were more expensive than they are today and a lot fewer people owned them than do today. During this period, there were a lot fewer home computers and not many were connected to the Internet, making the computers a resource that was not available to anybody but their owners. With the advances in networking technology, it became possible to enable multiple computers to collaborate on problems, thus distributing the work that a single computer would need to do to multiple computers and creating the paradigm of distributed computing.

When it became possible to use multiple computers to work on one problem, people began working on harnessing spare CPU cycles of machines in the labs of institutions with a set of networked computers, which lead to different types of distributed computing including networks of workstations (NOWs), the cycle-stealing paradigm, and the various forms of metacomputing. In the cycle-stealing paradigm, computers that are deemed to be idle are used to work on tasks, thus "stealing" the CPU cycles that would otherwise be not used for productive work. Condor is one of these cycle-stealing systems that allows people to submit tasks to a central computer and runs the tasks on idle computers in a group of computers [38]. Metacomputing is a type of computing where multiple computers connected by a network are used as if they were one parallel computer [39]. Metacomputing includes several different offshoots of distributed computing, including cluster computing, volunteer computing, and grid computing. Researchers have also worked on load balancing and load sharing to make metacomputing more effective.

2.1.1 Load Sharing and Load Balancing

One problem encountered in distributed computing is that some of the computers in a group of computers working together can be heavily loaded while others have very little to do. In order to remedy this problem, computer scientists developed load sharing and load balancing. Load sharing attempts to keep all computers busy by shifting tasks to the less busy computers, while load balancing tries to make the load on each computer in the group roughly equal. In load balancing, a process may be started on one computer and then halted and moved to another computer. In the simplest form, the process would be restarted from the beginning, but by using checkpointing, the state of the process may be saved on the computer it is running on, moved to another computer, and restarted from the checkpoint, losing potentially significantly less progress than it would if it was just restarted from the beginning on the new computer. Four policies are used to perform load balancing [40]. The *transfer policy* determines whether a task should be transferred. The *selection policy* determines which task should be transferred. The *location policy* determines the computer to which the task being transferred should be sent. The *information policy* determines how often to collect the system state data which is used by the other policies. The Satin [41] and ATLAS [42] systems explored a method of load sharing called *work stealing*. In the Satin system, clients have a pool of tasks. When a client has no more tasks in its pool, the client steals a task that is not being processed from another client's pool [41]. In ATLAS, threads are stolen [42].

2.1.2 Networks of Workstations

Networks of Workstations (NOWs) are typically composed of computers that sit on the desks of individuals and are primarily used by those individuals, and are connected by a network. Xu noted in 1996 that "a practical NOW system is heterogeneous and non-dedicated" [43]. In fact, a NOW composed of dedicated workstations would be called a cluster today. NOWs that are used for load sharing use the cycle stealing paradigm of distributed computing, allowing people to harness the processing capability of workstations that would otherwise be wasted. When a computer in a NOW becomes idle, it is given a task to work on until the user begins using the workstation again. Because in many companies all the employees have workstations that are significantly underutilized, the ability to use the computers productively when they would otherwise be idle is very valuable, allowing the computers to perform significant amounts of work when not in use. Although originally the software like Condor that assigned tasks to workstations in NOWs when they became idle only would ensure the task was stopped when a person started using the machine again [38], today some of the software used for NOWs runs tasks at a low priority. Running a task at low priority allows the user to do his or her work without detecting much slowdown, but also allows the workstation to be more productive, since many users do not use the full processing capability of their computers even when they are using the computers.

2.1.3 Cluster Computing

A cluster of computers also consists of a group of computers connected by a network. However, unlike NOWs, the computers in a cluster are not single computers that are each primarily used by different people. Instead, the computers in a cluster are typically maintained by a single person and are used for computationally intensive tasks that take too long or cannot be done on a single workstation. A heterogeneous cluster is composed of different types of computers, while a homogeneous cluster is composed of a single type of computers. Although the nodes in a cluster do not need to work on the same application as each other, the power of a cluster is the ability of the individual nodes to all work on a portion of a single application, resulting in performance that can be comparable to a supercomputer. Unlike an expensive supercomputer, a cluster can be composed of off-the-shelf computers, making them very cheap in comparison to supercomputers. Additional nodes can also be easily added to clusters, making them more flexible than supercomputers in terms of expandability.

2.1.4 Supercomputers

"A supercomputer is a computer that leads the world in terms of processing capacity, particularly speed of calculation, at the time of its introduction" [44]. Supercomputers can contain over one hundred thousand processors, terabytes of RAM, and special interconnect systems that allow the processors to communicate with each other. The quantity of hardware and the specially designed interconnects and cooling systems cause supercomputers to be extremely expensive. Similar to a cluster, a supercomputer's processors can all work together on one application that is decomposed into many smaller tasks. However, one of the major benefits of a supercomputer is that it is one computer to manage, unlike a cluster which is composed of many separate computers that must be managed. Some supercomputers have been built as one unit, while others are composed of many cabinets of equipment, analogous to the many different nodes that make up a cluster. Often supercomputers are used for very specific purposes, and it can be cost effective to have a supercomputer specially designed for a specific task like vector processing. Supercomputers that are custom designed for a specific task will have large performance advantages over an equivalent number of general purpose processors performing the same task.

2.1.5 Grid Computing

One of the newer offshoots of distributed computing is *grid computing*, a term Dr. Ian Foster coined during the latter half of the 1990's [45]. Foster's idea was that one would be able to get computing power the way one gets electricity – what one needs when they need it [45]. In *The Anatomy of the Grid*, Foster, Kesselman, and Teucke attempted to describe the "Grid problem," defining it as "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources" [46]. Foster later gave a set of three criteria that he felt captured whether a system fit the definition of *the Grid* and thus solved the *Grid problem* that he and his co-authors had published. The criteria are:

1. The resources used are not under the control of one entity, such as one company [47].

2. A grid uses “standard, open, general-purpose protocols and interfaces” [47].

3. A grid delivers “nontrivial qualities of service” [47].

In the meantime, the term “grid computing” has become a buzzword and been used to describe many types of computing other than Foster’s idea, causing a fair amount of confusion about what grid computing really is [45]. For example, many companies set up internal so-called grids and have the computers work on tasks such as drug discovery programs when they would otherwise be idle. Other companies set up these networks of computers and rent out the computational resources. Sun Microsystems currently has a network of thousands of computers and rents CPUs for \$1/hour each and disk space for \$1/GB per month [48]. These so-called grids are not really grids by Foster et al’s definition, but rather NOWs or clusters. However, Foster does note that the distributed systems that use computers from multiple sites such as Condor and the systems provided by Entropia, and United Devices could “reasonably be called (first-generation) Grids” [47]. Some grids that appear to fit Foster’s criteria do exist, including the Croatian Grid (CRO-GRID) and the South-East European Grid (SEE-GRID) [49, 50, 51].

2.1.6 Choosing a Distributed Computing Solution

We have presented several different paradigms of distributed computing that one might use for projects that cannot be completed in an acceptable amount of time by a single general purpose computer. It is important to select the appropriate tool for each task, however, to achieve the desired results within a project’s constraints. Supercomputers are best for projects with budgets of millions of dollars and the need for the processing power of hundreds of thousands of dedicated computers for a long time. NOWs are best suited to projects that are sponsored by people with access to NOWs and projects that are not time critical and do not need the power of a supercomputer. If a project requires faster turnaround than can be achieved by a NOW but not the amount of processing provided by a supercomputer and the sponsor has access to a cluster or can afford to purchase one, then a cluster would be a good tool for the project. In the case where a project needs the computational power of a supercomputer and the sponsor does not have the money to purchase one, then for projects that have some benefit to society, the volunteer computing paradigm is the appropriate one to use.

2.2 Volunteer Computing

Each volunteer computing project requires the computers that participate in the project to run a client program that is unique to the project. The difference between these clients is the algorithm, or *science application*, that the client uses to processes data received from the server [3]. However, aside from the science application, the remainder of the client for each volunteer computing project is essentially the same, and thus the clients all have similar requirements, aside from the science application. Therefore, a well-written client program could be used for many different volunteer computing projects by simply replacing the science application. The server programs for volunteer computing projects all have similar requirements. The similarity of the requirements means that a framework could be developed to facilitate the creation of volunteer computing projects. Such a framework could make project creation as simple as setting up a web page, selecting the values of some options in a GUI, and writing the code for the science application the clients run, the code to validate the results, and the code to generate tasks, as Baldassari showed in his M.S. Thesis [52]. We present a list of the components of volunteer computing projects and explain what they do. Then we discuss the major volunteer computing project frameworks. Finally, we discuss the design decisions that the creators of volunteer computing projects must make.

2.2.1 The Anatomy of a Volunteer Computing Project

Volunteer computing projects are developed using the client-server architectural model. The projects require both hardware and software on both the client and server sides. A project's sponsors provide the server hardware and both the server and client software, while the volunteers supply the client hardware. The bulk of the complexity of a volunteer computing project is in the server portion of the project. The hardware details are mundane and standard techniques such as using multiple servers and load sharing can be used, so we focus on the software that the server and clients run.

2.2.1.1 Server

The server has many responsibilities in volunteer computing projects. The server's first responsibility is managing the infrastructure for the project. This consists of hosting a web page where users can learn about the project and download the clients. The server must also run a database, which is used to store all the information about the progress of the project. In addition to this, tools such as scripts are needed to manage the database and coordinate all of the other functions of the server. The server must also provide facilities for the clients to download workunits. Also, the server needs to support any security required for the projects, such as secure communication and login names and passwords. The server must also manage and store the information about users, including how much work their clients have accomplished.

The next function of a server must perform is generating the tasks, also known as workunits, which the clients will perform. In the case of some projects, creating the workunits is simply chopping up large data files into small files, which are then used as input by the clients. In other cases, creating the workunits involves generating a lower and upper bound so that the client can search the values in between. Sometimes creating the workunits consists of identifying objects that the clients will then evaluate. When the server receives a request for a workunit from a client, the server should assign the client a workunit that the client is likely to be able to complete by the deadline associated with the workunit. Determining an appropriate workunit should just involve comparing the estimated computational power required to complete the workunit with the expected computational power of the client that will be available before the deadline. The client's expected computational power is calculated by benchmarks that the client runs and stored on the client.

The server usually performs some validation of the results that clients return by assigning each workunit to multiple clients and comparing the answers, in an attempt to keep faulty clients from participating and contaminating the results. Another step the server might take to prevent faulty clients from participating is *spot-checking* [12]. Spot-checking sends a workunit for which the server knows the correct result to a client and checks that the client returns the correct answer [12]. If a client fails a spot-check, the server might send it another spot-check or keep tabs on the client to ensure it is not faulty

[12]. The server may also stop sending workunits to a client that fails a spot-check if the server concludes that the client is faulty or corrupted.

2.2.1.2 Client

In contrast to the server portion of volunteer computing projects, the client portion has few responsibilities. Client programs typically perform the following jobs. When the client runs for the first time, a set of benchmarks is run on the computer the client resides on. These benchmarks indicate the computational power the client is likely to be able to contribute in some time interval. Then the client requests work from the server and receives a set of filenames, corresponding to workunits, to download from the server. The client downloads the files and then begins processing them, one at a time. To process a file, the client runs an algorithm, known as the science application. The science application performs some computations using the file as input and then returns the result to the server. At various points throughout the science application, checkpoints are created, saving the state of certain values so that if the science application needs to be restarted, it can start from the last known checkpoint instead of from the beginning of the science application. When the client does not have enough work to do, it requests more from the server.

2.2.2 Volunteer Computing Project Frameworks

Because the client portions of volunteer computing projects are so similar to each other and the server portions of volunteer computing projects are so similar to each other, it is possible to develop a set of tools, or a *framework*, for creating volunteer computing projects. A framework can simplify the job of creating a volunteer computing project by automating many of the tasks required to create a project and providing templates or skeletons for programs that must be written. There are several important volunteer computing frameworks, both past and current. The most influential of these frameworks are Bayanihan, BOINC, and Xtremweb, which were developed in academia. However, there are other proprietary frameworks as well.

2.2.2.1 Bayanihan

Luis Sarmenta developed Bayanihan at MIT as part of his Ph.D. dissertation [12]. Although no longer in use, "Bayanihan was one of the first general-purpose web-based volunteer computing systems using Java" and appears to be one of the first volunteer computing frameworks that was available. Bayanihan is a web-based volunteer computing framework that allowed users to volunteer their computer without fear of security risks or the need to install software. To participate, volunteers visited a web page with their web browser. The browser then automatically downloaded and ran a Java applet. Because the client for a Bayanihan-based project was written in Java, it enabled the creator of a volunteer computing project to write and maintain a single client and code base, simplifying the project sponsor's tasks.

2.2.2.2 The Berkeley Open Infrastructure for Network Computing (BOINC)

The Berkeley Open Infrastructure for Network Computing (BOINC) is one of the dominant frameworks currently used to build volunteer computing projects. BOINC was developed at the University of California at Berkeley in the Space Sciences Laboratory [3]. BOINC allows a person with a single computer (to act as the server) to create a volunteer computing project without writing all the networking code for a client and server. BOINC provides the scripts and tools to set up everything except the actual processing code, which the project creator supplies. BOINC allows the volunteers (participants) to customize several settings that affect how BOINC runs on their computers. The most important of these settings for our work is the one that allows volunteers to choose how much work the client buffers on their computers [54]. This setting directly affects the task retrieval method that we are studying.

2.2.2.3 Xtremweb

Xtremweb, developed at the University of Paris Sud, is another framework for creating volunteer computing projects [16]. Like BOINC, Xtremweb provides tools to assist in the creation of volunteer computing projects. In contrast to BOINC, Xtremweb only allows each client to retrieve one task at a time; when a client completes its assigned task and returns the result, it can request another task.

2.2.2.4 Other Miscellaneous Frameworks

In addition to frameworks like BOINC that are freely available for use, there are other software packages that can be used to create volunteer computing projects. Organizations such as Distributed.net that have created and run numerous volunteer computing projects do not likely code their projects from scratch each time, since very little needs to change between projects. Although Distributed.net may or may not have a formal framework, it is likely that they use the same code for each project except for the science application. There are also companies such as United Devices and (the now defunct) Entropia that produced programs that could be used for volunteer computing projects. These proprietary frameworks, not freely available, are intended to be used by companies such as pharmaceutical companies who want to utilize the spare computing cycles from their employees' desktop computers to test millions of combinations of chemical compounds, searching for new drugs.

2.2.3 Design Decisions for Volunteer Computing Projects

There are several design decisions that the creators of volunteer computing projects must make. Each of these decisions has an impact on the time it will take for a project to be completed. These decisions include determining the method clients use to retrieve tasks, the language used to write the clients, and the types of clients to provide. Other decisions that must be made that also affect the performance of the project are how often the server performs spot-checking and how many clients should receive the same task.

2.2.3.1 Task Retrieval in Volunteer Computing

Past and current volunteer computing projects have used various methods of how clients retrieve tasks. We focus on projects that require clients to download data files to perform assigned tasks, as opposed to a project like GIMPS where getting the data is trivial because participants only need to download a number. Currently there are several major file-based volunteer computing projects running from different organizations. Active projects include several projects that utilize BOINC [15], two projects from Distributed.net [53], and two projects from Grid.org [26].

When a client requests work, it requests an amount that will keep the computer busy for an amount of time that is configurable by the participant [54]. Thus, BOINC allows the user to determine how much data to send to the client, allowing a client to download many tasks at one time. The Distributed.net projects, like the BOINC-based projects, allow the participant to configure the size of the data set the client receives at one time [55]. As with BOINC, Distributed.net's method sends the user the amount of data that the user requests, which may result in the clients receiving too much data or too little data. As with BOINC-based projects, Distributed.net's projects allow clients to download multiple tasks at one time. For convenience, we refer to the method of retrieving tasks that both BOINC and Distributed.net use as the *Buffer N Days* method. Conversely, Grid.org and Xtremweb both have clients retrieve one task at a time, so a client will never download a second task until it has returned the results of the task on which it is working [16, 56]. For convenience, we refer to the method of retrieving tasks that both Grid.org and Xtremweb use as the *Buffer None* method.

Table 1 shows the framework used to create various volunteer computing projects and the task retrieval policies used by the projects.

Table 1 - Frameworks and Task Retrieval Policies of Volunteer Computing Projects

Project	Framework Used to Create the Project	Task Retrieval Policy
SETI@home	BOINC [2]	*Buffer N Days* [54]
Folding@home	none specified	*Buffer None* [57]
Einstein@home	BOINC [2]	*Buffer N Days* [54]
QMC@home	BOINC [2]	*Buffer N Days* [54]
LHC@home	BOINC [2]	*Buffer N Days* [54]
Rosetta@home	BOINC [2]	*Buffer N Days* [54]
Grid.org	none specified	*Buffer None* [56]
Climateprediction.net	BOINC [2]	*Buffer N Days* [54]
SIMAP	BOINC [2]	*Buffer N Days* [54]
The Riesel Sieve Project	BOINC [2]	Buffer Multiple [54]
World Community Grid	BOINC [2]	Buffer Multiple [54]

2.2.3.2 Volunteer Computing Client Implementations

Clients written in different languages provide a set of tradeoffs between efficiency, security, and convenience. These tradeoffs may influence the number of people that participate in volunteer computing projects and we discuss the tradeoffs in Section 3.3. Some clients, like the GIMPS client, are written in assembly language [58]. While writing clients in assembly language makes the client more efficient, it significantly increases the amount of work to port the client to different platforms. Other clients, like BOINC clients, are written in C++ or Fortran [3]. In addition to portability issues, the choice of languages currently influences the types of clients that projects can provide. We discuss the types of clients and their advantages and disadvantages in Section 3.2.3.4.

2.2.3.3 Spot Checking

As discussed in Section 2.2.1.1, spot-checking can be a low cost way to try to keep malicious participants from corrupting project results. However, since each check wastes the time it may take to do one real workunit, frequent spot-checks will slow down volunteer computing projects significantly. The project sponsors needs to determine the frequency of spot-checks that the sponsors feel will provide a balance between detecting malicious or faulty clients and delivering acceptable performance for the project.

2.2.3.4 Redundancy of Workunits

One decision that volunteer computing projects must make is how many clients to assign the same workunit. By assigning each workunit to more than one client, the server can potentially identify faulty returned values and provides some degree of assurance that the answer returned by the clients is correct, without having to validate each answer. It is important that the server be able to take all the results returned for a particular workunit and select what it deems is the *correct result*. Selecting the correct result is necessary because different clients may return different answers to the same workunit, due to hardware differences and malicious participants who try to cheat the system to get more credit than they deserve. Once the server selects the correct answer, it might notice that a particular client has given an answer that is very far from the one the server selected as correct. The server might keep a list of clients that return bad answers with some degree of frequency that is unacceptable and prevent those clients from participating in the future or double check some of the answers those clients have returned.

While assigning the same task to multiple clients allows the server to detect faulty clients and provides a measure of confidence that the project has the correct output from a task, doing so takes significant computing power that could otherwise be used to speed up the project. In some cases, it is very important to get the result of a task quickly, because the result may be used as input to a subsequent task. Assigning a task to more clients should decrease the time it takes to get enough results to determine the correct one, but it increases the amount of CPU cycles that clients spend on calculations that will not be used. Clearly there is a tradeoff between the certainty that a result is correct, the time it takes to achieve that certainty, and the overall completion time of a volunteer computing project.

2.3 Using Simulations to Evaluate Task Retrieval Policies

Although there were several techniques we could have used for evaluating task retrieval policies, we use simulations as our primary tool to evaluate task retrieval policies in this dissertation. We could also have used analytic modeling or implemented a volunteer computing project with different clients, each using a different task retrieval policy. We chose to use simulations because they have some significant benefits over the other two methods and the drawbacks to using simulations were more tolerable than the drawbacks of the other methods. We did also attempt to create an analytic model for the amount of work that a volunteer computing client would complete, based on the task retrieval policy it used, but we were unable to construct a useful analytic model.

There were several benefits to using simulations. Simulations yield easily reproducible results, while implementing an actual volunteer computing project would not yield results that could be reproduced, given the dependence on computer usage. Simulations also require much less cooperation from people than implementing a volunteer computing project would require. While simulations can easily be tuned and run again, one cannot easily ask the participants in a volunteer computing project to download a new client because we wanted to make some sort of modification to the client. There are some drawbacks to using simulations, rather than implementing a volunteer computing project. Simulations do not capture the low-level details that are captured by implementation. We omitted factors such as network congestion, packet delay, and the effects of operating system interactions with the volunteer computing clients in order to successfully develop the simulations. Thus, the results of simulations are not as accurate as the results of implementing a volunteer computing project would be. An advantage of using simulations instead of analytic models is that simulations can incorporate important real-world data, such as when a computer's screensaver is running. In contrast, analytic models must use distributions that may not accurately reflect actual data. Simulations can also incorporate more details than analytic models without becoming unwieldy. A drawback of using simulations is that they take a long time to run. While somebody with the values to use for the variables in an analytic model can do a few calculations and have an answer relatively quickly, it can take weeks of CPU time to run simulations. Simulations also have many places where mistakes can be made in the implementation, while an analytic model is much simpler. As we previously mentioned, we did attempt to create an analytic model. However, to try to make the model as useful and true to real life as possible, we found that we needed the analytic model to include information about the computers that would participate in volunteer computing and we were unable to incorporate this information into the analytic model in a simple enough way to keep the model from becoming unwieldy. Because of the drawbacks of analytic models and implementing a volunteer computing project, we chose to use simulations to analyze the task retrieval policies.

2.4 Summary

In this chapter, we discussed some of the areas of distributed computing, the anatomy of volunteer computing projects, several important volunteer computing project frameworks, and the design decisions that people creating volunteer computing projects must make. We have also explained why we use simulations to evaluate task retrieval policies and the benefits and drawbacks of using simulations for that task.

Volunteer computing is related to other topics in distributed computing, such as load sharing and load balancing, NOWs, clusters, supercomputers, and grid computing. Volunteer computing projects use the client-server architecture. The server portion of the project manages the infrastructure of the project, hosting a web page, running a database, maintaining participant information, and handling any security for the project. The server generates the tasks, assigns workunits to clients, validates the results from clients, and ensures clients are not corrupted. The client processes workunits and sends the results back to the server, checkpointing at intervals during the computations to avoid large amounts of work. Bayanihan, BOINC, and Xtremweb are three important volunteer computing frameworks. Bayanihan was one of the original ones but is no longer in use, while BOINC and Xtremweb are both in use, with BOINC being the dominant one. Project design decisions include how clients retrieve tasks, which language to use to implement the clients, the frequency of spot-checking, and how many clients must process a single workunit. Each of the design decisions has an impact on how long a project will take to complete. Although there are drawbacks to using simulations to evaluate task retrieval policies, the benefits of using simulations outweigh the drawbacks and thus, we use simulations to evaluate task retrieval policies.

3 Ways to Improve Project Completion Time and Tradeoffs

Because the uses for volunteer computing can have such important benefits for society, it is desirable to decrease the amount of time it takes to complete the projects that have a finite duration and be able to process more data in the same amount of time for projects with infinite durations. There are essentially two categories of methods, social and technological, that we can employ to decrease the amount of time to complete projects, assuming that we do not want to draw volunteers away from one project to complete another one faster. We do not want to draw volunteers away from one project to complete another project because that could slow down an important project and might result in projects with infinite durations not getting enough participants to make progress. Although this work focuses on the technological methods to decrease project completion time, we feel it is important to explain the social methods in enough detail so that others may work to address them. In addition to this, some of the social methods may influence which technological methods are implemented.

3.1 Social Methods – Recruiting More Participants

The social method to decrease the turnaround time of volunteer computing projects is to recruit more people to participate in volunteer computing. One way to do this is simply to increase the awareness of volunteer computing by increasing the publicity of volunteer computing. It is likely that some people who would participate in volunteer computing if they knew about it. For instance, families and friends of cancer victims might participate in a volunteer computing cancer research project such as Grid.org's project, in honor of loved ones who have succumbed to the disease. However, very few people I talk to are aware of volunteer computing projects. We suggest that some companies could dramatically help increase awareness of volunteer computing at no cost. For example, web browsers already come with a few links in the “bookmarks” or “favorites” sections. We would like to see web browsers come with a link to volunteer computing information and projects, such as the BOINC web site. Internet portals such as Yahoo and Google could also put links on their web site which could help increase awareness. Google did have a toolbar feature that allowed users to contribute to Folding@home, a volunteer computing project at Stanford, but that is no longer available. Companies that produce operating systems could also bundle information about volunteer computing or even volunteer computing clients themselves with the operating systems. We note that BOINC became part of the Debian Linux distribution in April 2006 [59].

Although increasing the publicity of volunteer computing projects may help recruit more participants, there are still people that are aware of volunteer computing who choose not to participate for a variety of reasons. One reason people do not participate is that they are afraid that participating may cost them too much [60]. The cost of electricity to run a computer constantly or the additional electricity required to run the computer’s components instead of having them doing nothing while the computer is on but idle is one deterrent [60]. Other people are concerned that the extra wear and tear to their computer will cause it to fail sooner [61]. Some people are afraid that participating in volunteer computing projects will put their computer and personal data on the computer at risk [60]. The problem of putting personal data at risk can be solved with technological methods and additional education of the public. Some people are concerned that running a volunteer computing client will slow down their other programs. Properly written clients need not slow down users' computers noticeably, so additional education of the public and having a third party verify that the client was written properly could resolve this issue. Another way to prevent people from worrying that the client will slow down their computers is to run the client only when the user is not using their computer, which can be done by constructing the client as a screensaver. Many people seem comfortable with the idea that a screensaver only runs while they are not using their computers actively. Another obstacle that prevents users from participating in volunteer computing is computer illiteracy. Some users are uncomfortable using their computers to perform mundane tasks like downloading or installing programs. If the obstacle of downloading and installing the program can be removed, more of these users may participate in volunteer computing. The obstacles of downloading or installing clients can be mitigated with a technological solution. A web-based volunteer computing client, such as a Java applet, does not require users to explicitly download or install the client.

A final way to increase participation may be to offer incentives to participants. Many existing volunteer computing projects give users "credit" for the amount of work their computers have contributed. People compete as individuals and on teams to rack up more credit than others [1]. The SETI@home project even offers printable certificates to users who have completed various amounts of work for the project [62]. However, these incentives are likely not enough to convince some people to participate in volunteer computing. We suggest that financial incentives may be sufficient to increase the number of people who participate in volunteer computing and could potentially convince companies to participate in volunteer computing. This could increase the number of volunteered computers significantly. One purpose of volunteer computing is to allow research to be done without a significant outlay of cash, so it is not reasonable to assume that all of the organizations that sponsor volunteer computing projects could provide financial incentives. However, governments could offer tax deductions or credits and other organizations such as charitable organizations or businesses could provide prizes or discount coupons as incentives [63].

3.2 Technical Methods

Volunteer computing projects make use of hardware and software. Improving the performance of the hardware that volunteer computing programs run on and the performance of software, like databases, that volunteer computing projects use would improve the completion time for volunteer computing projects. However, teams of hardware and software designers work to improve hardware and software performance, so it is clearly beyond the scope of our work to try to improve hardware and database performance. Porting the clients to additional devices could potentially increase the number of devices participating and we discuss this in Section 3.2.1. The remaining technical methods for improving the completion time for volunteer computing projects must be implemented in the software used by volunteer computing. The volunteer computing software consists of the server program, the client, and the science application. Algorithmic improvements to the science applications that the clients run would also improve the completion time for projects, but this too is beyond the scope of our work and we leave it to others. It is not feasible to make algorithmic improvements to the science applications because doing so would require an in-depth analysis of the science applications. However, the source code for most of the science applications is not available and understanding many of the science applications may require domain specific knowledge from fields other than computer science. In addition, as the number of projects increase, there would be more science applications to analyze. The time consuming task of analyzing science applications would only benefit the projects examined, as opposed to all projects, and some science applications may already be optimized and would therefore, not benefit at all. Thus, the areas we focus on are porting clients to additional devices and modifications to the server and client programs.

3.2.1 Porting the Client to Additional Devices

Due to the computational demands of volunteer computing projects, it is desirable to find additional sources of volunteer computing power. We explored this idea in [64] and the information in Section 3.2.1 comes from that work. In order to be a viable source of volunteer computing power, a platform must be able to provide enough CPU cycles to make it worth the effort to port volunteer computing applications to that platform. General-purpose computers are the most powerful devices used for volunteer computing (nobody volunteers their supercomputer) and it takes many of them to make progress on a project. Therefore, it is clear that a platform that is not as powerful as a general purpose computer must have even more units in circulation to produce enough computational power to make progress on volunteer computing projects. Some devices that have the computational ability to perform calculations used in volunteer computing projects are cell phones and PDAs. However, because performing the calculations for volunteer computing projects is so CPU intensive, running a client on a mobile device such as cell phones will drain the device's battery very quickly. Since people will not want to drain the batteries of their cell phones or PDAs in minutes, it does not make sense to port a volunteer computing client to mobile devices. However, video game consoles do not run on batteries and millions are sold. By the end of 2005, Sony had sold 200 million PlayStation2 video game console systems [65]. Video game consoles have become increasingly powerful computers over the last 30 years. The Atari 2600, released in 1977, had a 1.19 MHz CPU and 128 bytes of RAM [66]. Sony's PlayStation2 contains a CPU running at 299 MHz, 32 MB of RAM, an optional Ethernet connector and custom graphics hardware [67, 68]. Microsoft's Xbox has a 733 MHz Intel processor, 64 MB of RAM, an Ethernet connection, and specialized graphics hardware [69]. Microsoft's Xbox360, the newest video game console system, was released in the Fall of 2005. This console contains a triple core CPU with each core running at 3.2 GHz, 512 MB of RAM, an Ethernet connection, and specialized graphics hardware [70]. The number of video game consoles sold and their computational power combined with their network capability makes them a potentially good platform for volunteer computing.

3.2.1.1 Testing the Viability of Video Game Consoles for Volunteer Computing

We wanted to understand how viable a platform for volunteer computing video game consoles are, so we devised an experiment to test console systems and compare the results to the results from computers.

3.2.1.1.1 Hardware

To test the viability of video game consoles for volunteer computing, we conducted experiments using a server, various computers running a client, and a 10 Mbps Ethernet hub. The server had an Intel Pentium III 450 MHz processor and 256 MB of RAM. It ran Windows 2000 with Service Pack 3 and the Apache web server version 1.3.34. We used a variety of different computers to run a volunteer computing client. We used computers with the following CPU and RAM configurations to run a volunteer computing client:

- Pentium II 233 MHz processor and 320 MB of RAM (general-purpose computer)
- Celeron 400 MHz processor and 128 MB of RAM (general-purpose computer)
- Pentium III 733 MHz processor and 128 MB of RAM (general-purpose computer)
- Pentium IV processor and 512 MB of RAM (general-purpose computer)
- 299 MHz processor and 32 MB of RAM (PlayStation 2 video game console).

Because only the PlayStation 2 had a programming environment available for entities other than game development companies at the time of the tests, we were only able to develop tests for that video game console.

3.2.1.1.2 Software

We developed a volunteer computing client to enable it to run on the general-purpose computers and the PlayStation 2. The client performs fast Fourier transforms, like the SETI@home client does [71, 72]. We had developed the client like this because unlike most volunteer computing projects, the source code for SETI@home is available, allowing us to see what a volunteer computing client does. To save time, we used freely available fast Fourier transform code for our client [73, 74]. Like other volunteer computing clients, our client downloads a task from a server to work on, performs the computations, occasionally checkpointing to disk to avoid losing all the progress in the event of a system failure, and reports the result of the task back to the server.

3.2.1.1.3 Experiments

We ran nine tests on each computer and game console, one test corresponding to each parameter combination. Since we had not yet fully explored the parameters that existing volunteer computing projects use (see Section 6.1), we used what we considered reasonable values for these parameters in this study. The parameters were file size (16 KB, 512 KB, and 1 MB) and computation intensity (performing the task 1, 2, or 4 times). For each test, the clients requested a task from the server, downloaded a file to be used in the task from the server, saved the file to the local disk, ran a computation on the contents of the file, and returned the result of the computation to the server 25 times. This was repeated 10 times. In between the tests from the different computers, the web server was rebooted to ensure that the cache was in the same state for the tests and the computer the client ran on was also rebooted.

3.2.1.1.3.1 Head to Head Comparison

The results of our experiment are shown in Figure 1. The PlayStation2 was significantly outperformed by all the computers we tested. We were not surprised by most of the computers outperforming the PlayStation2 significantly, except for the Pentium II 233 MHz computer. This computer was the closest comparison to the PlayStation2 that we had, and just based on CPU speed, we expected the PlayStation would perform roughly the same as the computer. The computer had significantly more RAM (320 MB compared to the Playstation2's 32 MB), so we hypothesized that the computer was keeping data in RAM that the PlayStation2 had to swap out to the disk. Therefore, we tested the computer again with only 32 MB of RAM in it. The results were almost identical to the ones we obtained when the computer had 320 MB of RAM. Thus, it is unlikely that the small amount of RAM in the PlayStation2 is causing its slower than expected performance relative to the Pentium II 233 MHz computer.

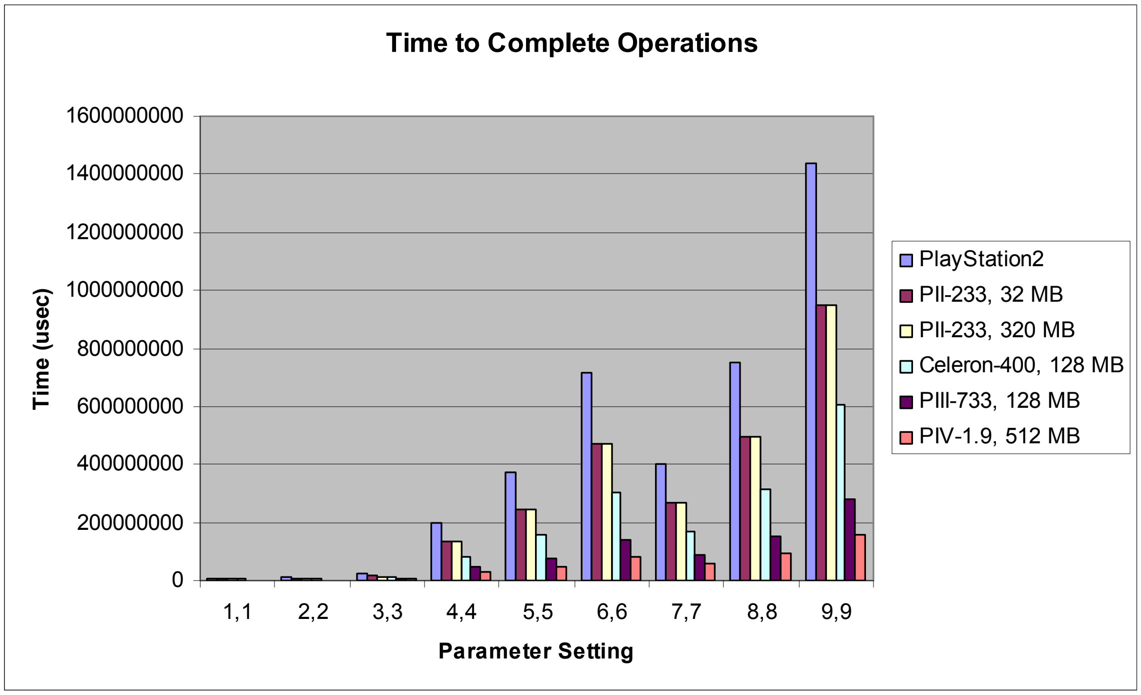


Figure 1 - Volunteer Computing Client Performance on Various Hardware Platforms

3.2.1.1.3.2 Processor-Only Comparison

The hardware for the PlayStation2 is specialized and the hard drive and network interface adapter are connected in a very different manner than those of normal computers. The hard drive is plugged into an expansion port and the network interface is plugged into the back of the hard drive. Thus any network traffic must pass through the physical casing of the hard drive. We believed that the differences between the hardware connections of the PlayStation2 and personal computers might account for some of the performance differences we observed. In order to test this, we modified our test program to eliminate both the network and disk I/O. We placed all the data that normally needed to be retrieved from the server, saved to disk, and then read in from the disk in memory within the client program and recompiled it. Running the modified client on the PlayStation2 and the computer with the Pentium-II 233 MHz processor and 32 MB of RAM produced results that were even more puzzling. The computer with the Pentium II-233 MHz processor outperformed the PlayStation2 for this program by a significantly greater margin. Therefore, we believe that the PlayStation2's unique hardware connections are not likely a significant cause of any performance differences. We note that the PlayStation2 uses the ext2 file system, as opposed to the other computers we tested which use the ext3 files system. The ext3 file system is a journaling file system, while the ext2 system is not. Thus, the ext2 system may have given the PlayStation2 a little performance edge over the general-purpose computers. Therefore, by removing the file system usage, the program may have performed worse on the PlayStation2 than before compared to how it performed on the general-purpose computers. It is also possible that the cache in the general-purpose computer gave it a very large advantage over the PlayStation2. To test this, we would need to devise some micro benchmarks and run them.

3.2.1.1.4 Conclusions

The PlayStation2 was significantly outperformed by the general purpose computers, even the one with a CPU with a slower clock speed. In addition to this, we note that video game console systems are not often left on while they are not being used, which could hinder their usefulness in volunteer computing by limiting the number of CPU cycles that the systems might contribute. Because of these shortcomings, it is clear that the PlayStation2 is not the best resource for volunteer computing (clearly general-purpose computers are better). Despite these shortcomings, it is important to note that the PlayStation2 was able to perform the required operations for volunteer computing, including network and file I/O. Thus, we believe that newer faster game consoles such as the Microsoft Xbox360 Sony PlayStation3 that have about five years of hardware maturity over the PlayStation2 may be much more useful for volunteer computing. In fact, unbeknownst to us, as we were testing the PlayStation2, Sony and Folding at Home were collaborating to create a volunteer computing client for the Folding at Home project that would run on Sony's (then upcoming) PlayStation3 game console [75]. The client is integrated in the PlayStation3 and utilizes the PlayStation3's advanced hardware and new workunits created for PlayStation3 clients. The PlayStation3 is able to complete one of these workunits in 8 hours, so running the client overnight while the console is idle will complete a workunit. Folding at Home states that 50,000 PlayStation3 consoles would be able to attain "performance on the petaflop scale" [75]. The work from Sony and Folding at Home suggests that state-of-the-art video game consoles could be useful for volunteer computing.

3.2.2 Server

In this section, we analyze the server portion of volunteer computing projects and discuss the performance effects for volunteer computing projects that could result from making changes to the server. The server for a volunteer computing project has several different functions, as discussed in Section 2.2.1.1. We first discuss the four critical functions of the server that could cause it to be a bottleneck in a volunteer computing project. Then we explain the impacts that improving other parts of the server program would have on the project completion time.

The first critical function of the server to minimize the duration of a project is creating tasks that clients will retrieve. Creating tasks is important because if clients are available to process tasks and there is nothing for them to do, this wastes volunteered CPU time and slows down the project's completion. We are unaware that the creation of tasks is currently a problem, but if it was a problem, additional computers could be added to increase the amount of tasks created.

The second function of the server is allowing clients to download tasks when the client is ready and not act as a bottleneck for clients trying to download tasks. If clients are unable to get tasks quickly, then some of the time they could spend working on the tasks will be wasted and this clearly slows down projects. Anderson et al showed that one computer that cost about $4000 in 2005 could support clients retrieving 8.8 million tasks per day and that adding two more identical computers increased that to 23.6 million tasks per day [76]. Clearly additional task servers can be added to avoid this becoming a bottleneck.

The server must also ensure that tasks do not get sent to clients that are unlikely to return the results in time for them to be useful. The BOINC framework runs benchmarks to avoid sending tasks to computers that will be unlikely to complete them in time so we do need feel that this area needs further work currently [77, 78].

The fourth critical job for the server is sending tasks to enough computers to ensure that enough results are received to allow the server to select the correct result, but sending each workunit to as few computers as possible to avoid wasting volunteered CPU time. Because each project has its own requirements for how many computers it wants to assign an individual task to so it can decide on the correct answer, we do not feel it is necessary or valuable to explore this area.

The remaining functions of the server such as validation, spot-checking, and database management will not impact the project's completion time significantly if they are speeded up or slowed down. Although freeing up the CPU of the server could allow it to spend the idle time processing tasks it would otherwise assign to clients, this would result in such a small effect on the project that it would not practically affect the project's completion time.

3.2.3 Client

There are several aspects of volunteer computing clients that could potentially be improved, leading to decreasing the time volunteer computing projects take. It is important to note that the task retrieval policy for volunteer computing projects is implemented in the client, so we discuss this aspect of volunteer computing projects as if it is part of the client, even though it is really the volunteer computing project's task retrieval policy. Whenever a client asks for work, the server gives it some. Improving task retrieval methods, security, ease of installing the client, and providing multiple types of clients might all increase the effectiveness of volunteer computing.

3.2.3.1 Task Retrieval Policies

The information from Section 3.2.3.1 comes from our preliminary research, published in [79], with some minor modifications. We point out that the results in this section are from our preliminary work and they suggested that the percent of donated CPU time wasted by volunteer computing clients using different task retrieval policies was significant in some cases. There was a clear difference in the percent of time wasted by the different task retrieval policies in some cases. That difference was what motivated us to continue this work. If there had not been a significant difference, it would not have been logical to continue this work.

3.2.3.1.1 Comparison of Task Retrieval Methods

In preliminary work, we investigated the relative strengths of different methods clients may use to retrieve tasks. We examined three methods: the one used by BOINC and Distributed.net, the one used by Grid.org and Xtremweb, and one of our own design to provide another method for comparison. The method used by BOINC and Distributed.net, which we refer to as *Buffer Multiple*, has clients download multiple tasks at one time and buffer them. We consider the *Buffer Multiple* method with the various amounts of hours buffered as different methods for our measurements, and we call the method *Buffer-x* where *x* is the number of hours buffered. The method used by Grid.org and Xtremweb, which we call *Buffer None*, has a client download only one task at a time, downloading a new task only after returning the result of the previous task. This method guarantees a client does not buffer any tasks. After considering the two methods in use by volunteer computing projects, we concluded that all of the benefits that could be derived from buffering tasks could be gained by only buffering one task. Thus, we created a method to compare to the methods currently in use. The method we designed, called *Buffer1*, has a client buffer exactly one task by downloading a task while it is executing its current task. Using this method as well as *Buffer Multiple*, whenever the client finishes a task, there is another one waiting for it. We acknowledge that if downloading a task takes longer than it takes for the client to complete a task, then it would be possible that a task might not be waiting in the buffer and there would be wasted time waiting for the download to complete. However, we assume that files are small (10 MB or less), download speeds are fast (300 kbps or faster), tasks take a long time to complete (24 hours of CPU time), and the computers are constantly connected to the Internet. Based on our assumptions, we note that downloading a task should always take significantly less time to complete than performing a task and therefore, a client that buffers tasks will always have a task available to process. Both the *Buffer1* and *Buffer Multiple* methods eliminate the possibility of a client having idle cycles while it downloads another task, which occurs with the Grid.org method. However, buffering tasks increases the likelihood of tasks being late and having to be aborted. Buffering fewer tasks should decrease the likelihood of tasks being aborted due to an interruption of the client resulting from a system failure, increased system usage, or any other reason. We compare how much time each of the different methods of retrieving tasks wastes.

3.2.3.1.2 Comparing Wasted Time for the Systems

In order to compare the wasted time incurred by the different methods clients use to retrieve tasks, we constructed simulation models of these retrieval methods. However, we needed to be able to determine the wasted time caused by the different methods of retrieving tasks in order to collect data from the simulations.

We analyzed the different methods that the various projects' clients use to retrieve tasks to determine the time the clients waste. We then derived an equation for this wasted time and found that the wasted time is

- the time the client spent on idle cycles while waiting for a task to download (W)
- the time the client spent downloading files for tasks that the client does not complete on time (D)
- the time the client spent on working on tasks that are not completed on time (U). This does not include the time to download the files for the tasks.

For each method,

$$\text{Wasted time} = D + U + W \qquad (1)$$

We note that for BOINC based projects, Distributed.net projects, and if there were projects using the *Buffer1* method, W = 0 because at least one task is always downloaded before the last one in the buffer is completed. Because there is always one at least task in the buffer, the CPU is not forced to sit idle while waiting for a task to be downloaded. In contrast to this, for Grid.org and Xtremweb projects, W = the sum of the idle cycles which occur during the downloading of tasks because the parallelism of modern computers enables them to perform other instructions while downloading a file instead of the downloading occupying the CPU completely.

3.2.3.1.3 Simulations

3.2.3.1.3.1 Simulation Design

Using Equation 1 from our model in Section 3.2.3.1.2, we developed a simulation to gather data about the amount of wasted time incurred by the different methods of retrieving tasks. The simulation assumed that the size of the tasks would allow the clients to complete the tasks barring lengthy interruptions. The simulation involved many variables such as the time between the end of an interruption and the time that the next interruption occurs, which we refer to as an *available period* since the computer is available to work on volunteer computing tasks. Another variable is the duration of the interruptions that the client experienced, which we refer to as *unavailable periods*. During *unavailable periods*, we assumed that the client is unable to make progress on the task and that the task is simply paused but can be resumed in the state it was in when the task was paused. Tasks were started based on the order in which they were downloaded.

The set of simulation parameters was:

- File Size. The size of the file required by each task, that needed to be downloaded before starting a task (in MB).
- Download Speed. The client's download speed (in bps).

- Completion Time. The time it takes to complete a task (in hours) if it is not interrupted. It is assumed that all tasks take the same amount of time, specified by this parameter, to complete.
- Buffered Hours. The number of hours of work a BOINC client buffers (not relevant for the *Buffer1*and *Buffer None* methods). If the number of hours of work to buffer was not a multiple of the time required to complete a task, the number of tasks buffered was equal to the maximum of 1 and the floor of (hours buffered divided by the time to complete a task).
- Retrieval Method. The method clients use to retrieve tasks. We tested the *Buffer None*, *Buffer1*, and *Buffer Multiple* policies that we identified in Section 3.2.3.1.1.
- Abort Multiple. This is an indication of when a task should be aborted. If the abort multiple is a and the normal time to complete a task is t hours, then the task will be aborted if it is not completed within $a \cdot t$ hours after the expected start time of the task. The expected start time of a task is the time when the task is downloaded plus (the number of tasks ahead of it multiplied by the completion time of a task). The expected start time of a task is determined when the task is downloaded and does not change.
- Length of available periods. The time between the end of an interruption and the beginning of the next interruption (in hours).
- Length of unavailable periods. The duration of the interruptions (in hours).

3.2.3.1.3.2 Simulation Development

We wrote a program in Java to perform the simulations. We did 1000 runs and calculated the estimated variance. From the estimated variance, we determined that 150 runs gave a reasonable half-width for the confidence interval and we determined that 150 replications of the simulations would be sufficient. The simulations were run for 150 replications with each combination of the simulation parameters specified in Section 3.2.3.1.3.1. Each replication simulated a volunteer computing client for a period of 2000 hours where time is divided into units of one second. We assumed exponential distributions for the interruption times and durations and we used a library from the ARMiner project [80] to generate the random numbers we needed for the distributions. Because this was a preliminary study, we intended to replace the exponential distribution with a more realistic one after we collected data (see Chapter 4). For each combination of simulation parameters, the random number generators were seeded with the same seed to ensure consistency between parameter combinations. Thus, the i^{th} replication of a simulation for each parameter set used seed s_i. During the simulation, when an interruption occurred, the task being executed was paused until the interruption was completed. Once an interruption was completed, the computer resumed working on the task.

3.2.3.1.4 Results

We ran our simulation using the download speed of 300,000 bps, a file size of one MB, and a task completion time of 24 hours. We varied the abort multiple and the task retrieval method, as those are the factors that the creators of volunteer computing projects can control, and thus knowing the effects they have on waste should be useful to project developers. The first set of simulations assumed that the length of *available periods* was exponentially distributed with a mean of eight hours and the length of *unavailable periods* was exponentially distributed with a mean of four hours. Figure 2 shows the percent of time wasted under those conditions.

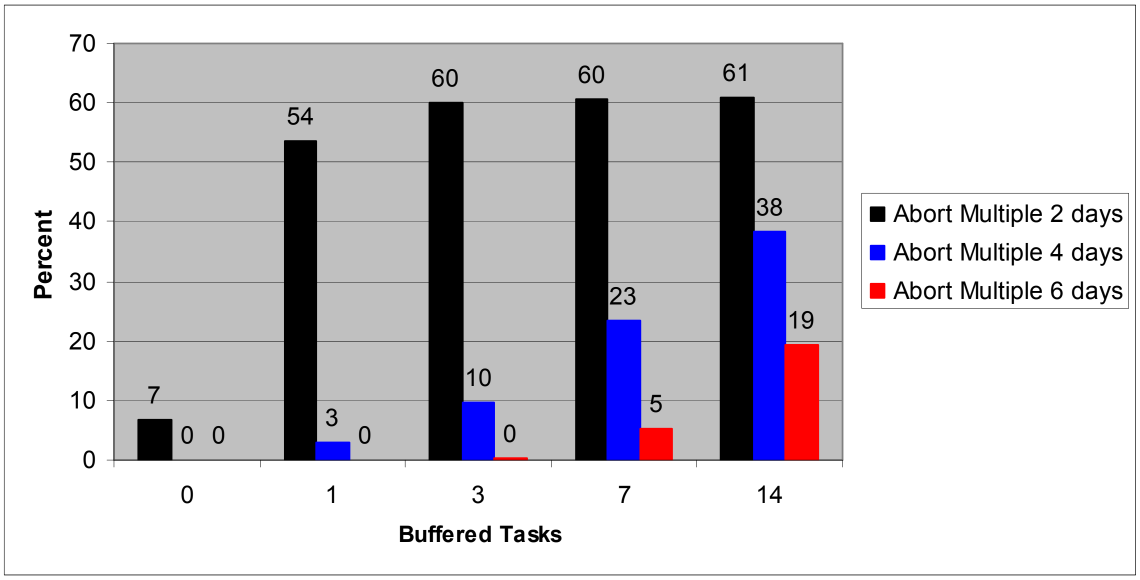


Figure 2 - Time Wasted For Mean *Available Period* Eight Hours and Mean *Unavailable Period* Four Hours

The *Buffer None* method (corresponding to the zero Buffered Tasks on the horizontal axis in Figure 2) wasted significantly less time than the other methods with the abort multiple two. When the abort multiple was four, *Buffer None* still wasted far less time than *Buffer*-72 (three buffered tasks), *Buffer*-168 (seven buffered tasks), and *Buffer*-336 (14 buffered tasks), but only a little less than *Buffer1* and *Buffer*-24. *Buffer*-24 wasted the same amount of time as *Buffer1* for abort multiples two and four. However, when the abort multiple was increased to six, *Buffer1*, *Buffer*-24, and *Buffer*-72 were as efficient as *Buffer None*. Increasing the abort multiple decreased the amount of wasted time for all the methods of retrieving tasks, and as the number of hours buffered by a *Buffer Multiple* client increased, so did the amount of time the client wasted. We duplicated the simulations with download speeds of 28,000 bps, 1,000,000 bps, and 10,000,000 bps and noticed almost no difference in the results.

For significantly less frequent and shorter interruptions, the results were very different. Our second set of simulations assumed the duration of the *available periods* was exponentially distributed with a mean of 16 hours and the duration of the *unavailable periods* was exponentially distributed with a mean of two hours. We varied the number of tasks buffered (zero, one, three, seven, and 14) and the abort multiple (two, four ,and six). Figure 3 shows the results of the simulation with those parameters; the number of buffered tasks is shown on the x-axis and the results based on the different values of the abort multiple are indicated by the different colored bars. For several of these parameter settings, the wasted time was zero.

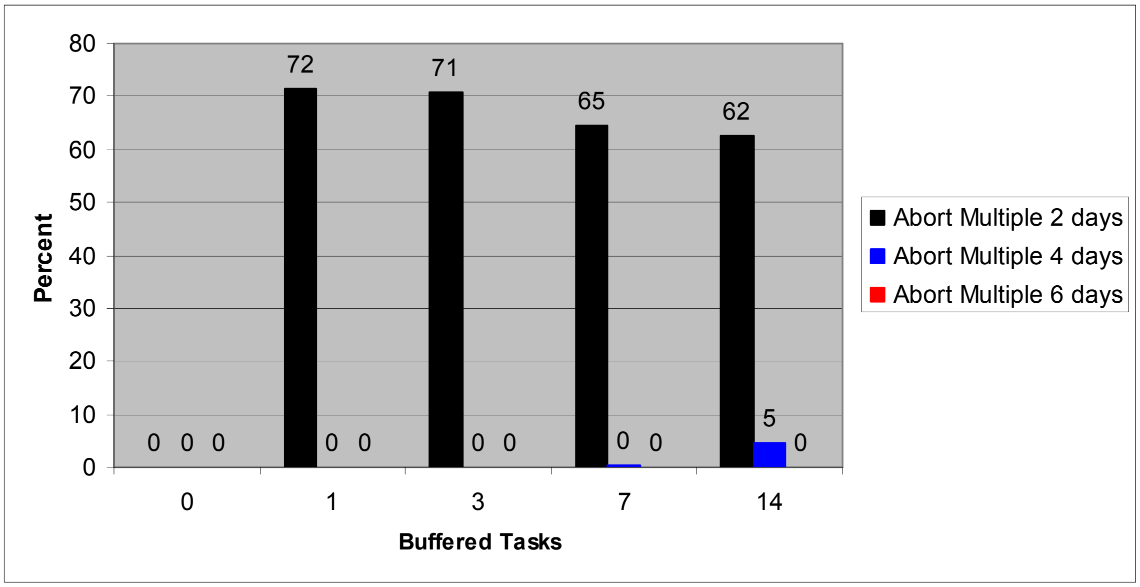


Figure 3 - Time Wasted For Mean *Available Period* 16 Hours and Mean *Unavailable Period* Two Hours

Although when the abort multiple was two, the *Buffer None* method was significantly better than the other methods, when the abort multiple was four or six, there was almost no difference between the amount of time wasted by the different methods. With the exception of *Buffer*-168 and *Buffer*-336 for the abort multiple four, all the methods wasted almost no time. Once again, there was almost no difference in the results for download speeds of 28,000 bps, 300,000 bps, 1,000,000 bps, and 10,000,000 bps. Our simulation also showed us that the values of several parameters had little or no effect on the results of the simulation. The values of other parameters had a significant impact on the results, causing the three retrieval methods to waste about the same amount of time. We tested file sizes of one, five, and 10 MB and download speeds of 56,000, 300,000, 1,000,000, and 10,000,000 bps. The different values of these parameters had almost no impact on the simulation results. We tested the values one, 12, 24, 36, and 48 hours for the time required to complete a task. There was little difference between the amount of time the different retrieval methods wasted for 1 hour tasks. However, the *Buffer None* method wasted far less time for the other task completion times. We also ran simulations with abort multiples of one and eight, which resulted in almost identical amounts of wasted time for the different methods of retrieving tasks.

3.2.3.1.5 Analysis

The results of our simulation have shown two important points. The results showed that a client retrieving one task at a time instead of retrieving multiple tasks at once causes less wasted time. In some cases, the amount of wasted time is significantly less using this method than using the others. In addition, while lower abort multiples cause significantly more wasted time for all the task retrieval methods, the effect of a low abort multiple on the *Buffer None* method causes far less wasted time than it does on the other task retrieval methods. We do point out that the parameters we used for the simulations were only "best guess" values and we ran the simulations again with actual values, as discussed in Chapter 4, Chapter 5, and Chapter 6. We also note that we assumed that tasks were simply paused during *unavailable periods*. In some cases, however, unavailable periods may be due to a computer being powered off, which would mean a task would need to be restarted from its last checkpoint when the computer became available to perform volunteer computing tasks again.

3.2.3.2 Security

The concern that many people have about their computer being compromised or personal data being stolen can be alleviated by building a client that does not allow access to a computer's file system and only allows network connections back to the volunteer computing project server. An unsigned Java applet cannot access the file system on a computer it is run on and can only make network connections to the server it was downloaded from [81]. Therefore, by writing a client as an unsigned Java applet and explaining that it is secure to people who do not understand about applets, projects could potentially increase their number of participants. We note that other similar methods could be constructed too using other programming languages, but we are currently unaware of other implementations.

3.2.3.3 Ease of Installation

The ease of installation issue can be resolved by constructing a Web-based volunteer computing client that runs when a Web browser visits a web site. Again, one implementation that allows a user to run a client without installing the client is a Java applet, although other Web-based applications could certainly be constructed.

3.2.3.4 Providing Multiple Types of Clients

One method that might improve the completion time of volunteer computing projects is for projects to provide multiple types of volunteer computing clients. The different types of clients might have different features, such as added security, that make them more attractive to different users. Therefore, a person who is not willing to run one type of client and would not participate if no other client was available might decide that the features of another type of client were acceptable and run it, thereby increasing the number of users. In the following sections, we discuss several different types of clients:

programs that run when certain conditions are met, programs that run all the time, and programs that run when a user explicitly activates them.

3.2.3.4.1 Programs That Run When Certain Conditions Are Met

One type of client that can be provided is a program that runs only under certain well defined conditions, such as every Saturday and Sunday or after a some specified amount of an absence of user input from the keyboard and mouse. We note that the different possible implementations of a program that runs when certain conditions are met are all very similar and the advantages and disadvantages between the various implementations are the same. Thus, we continue this explanation in the context of one such implementation which volunteer computing projects provide, a screensaver. In the case of the screensaver, the conditions that must be met are a lack of input from the keyboard and mouse for a specified time period. When the user's screensaver would normally be activated, the volunteer computing client is run and it displays its own graphics, which serve as a screensaver. The graphics usually relate to the project. For example, the Grid.org cancer research project displays information about the molecules it is currently testing, while the SETI@home screensaver displays information about the signal the client is processing. One advantage of clients that run when certain conditions are met is that after the program has been configured the first time, the user never needs to activate it explicitly. Another advantage of this type of client is that it does not run when the user does not want it to run. In the case of a screensaver implementation of this type of client, the screensaver does not run when a person is actively using their computer, so users should not be concerned that the client is slowing down their other programs when they are using their computer. A disadvantage of this type of client is the need to install it on the computer. Although the installation process of volunteer computing clients is both quick and simple, many potential volunteers may not be comfortable installing it. Another disadvantage of this type of client is that it provides less donated CPU time to volunteer computing projects than a client that runs all the time when a computer is on.

3.2.3.4.2 Programs That Run All the Time

A second type of client that volunteer computing projects can provide is a program that runs all the time while a computer is powered on. This type of client can be implemented as an application that the user invokes manually when he starts the computer or it can be set to run automatically on startup like a daemon process on Linux and Unix or a service on Windows. For simplicity, we will refer to the implementation that starts automatically as a daemon, regardless of the operating system it runs on. The application or daemon can run at a minimum priority level so that it does not noticeably interfere with the other programs when a person is actively using the computer. However, unlike a screensaver, the process uses the idle cycles on the computer even while a person is actively using the computer. An advantage of this type of program is that such an application or daemon process will provide more computational power to a volunteer computing project than a program that runs only when certain conditions are met. Since applications can be set to launch automatically when a computer starts and since daemons do not need to be activated explicitly, we consider both daemons and applications as having the advantage of not needing to be activated explicitly. Like a program that runs when certain conditions are met, a program that runs all the time suffers from the disadvantage of needing to be installed. In addition to this, in contrast to a program that only runs when certain conditions are met, some people are suspicious that a daemon will slow down the response time of their other applications noticeably.

3.2.3.4.3 Programs That Run When the User Tells Them To

A third type of client that volunteer computing projects can provide is a program that runs when the user "tells it to" by activating it and stops running when the user deactivates it. This type of client could be implemented as an application or as a web-based program. One advantage of this model is that users have complete control over when the application is running, so they know it is not slowing down their other programs when they deactivate it. A web-based implementation of the program would allow the user to run the program without installing it, but by simply visiting a web site and clicking a button to start (or stop) the program. The user could even set the site as their home page with the click of a button so every time they opened their web browser, they might remember to start the program if they wanted to. There are several drawbacks to this model, however. An application implementation would require downloading the client. It is also likely that many people participating by using this type of client, regardless of the implementation, would often forget to start it and thus a lot of CPU time that could be donated would be wasted. In addition to this, many people might decide it is inconvenient to keep starting and stopping the program, which could lead to them not participating.

3.3 Tradeoffs

There are tradeoffs that must be considered when evaluating the different types of volunteer computing clients. At a high level, clients that run constantly clearly provide more donated CPU cycles than clients that run when a user tells them to or when certain conditions are met. However, this must be weighed against the fact that many people do not want the clients to run when they are actively using the computer, and thus releasing only clients that run all the time may decrease participation. Constructing all three types of clients would certainly create the greatest number of participants, but it may also require several different programs to be written and maintained, which increases the burden on an organization running the volunteer computing project. In addition, some people who might have been willing to run a client that runs all the time may choose to run a client that runs only under certain conditions or only when the user instructs it to. Making all three types of clients available could result in the contributed amount of cycles from some users going down if users that might be willing to run a client that runs all the time choose to run a client that does not.

Implementing the client as a web-based application can eliminate the need for the user to download and install the client. However, web-based clients shift the burden of activating the client to the user. By leaving the barrier of requiring the user to download and install a client, the burden of activating the client can be removed from the user.

There are also some implementation specific tradeoffs that should be examined. The tradeoffs are the result of different features of currently popular programming languages like C, C++, and Java. The tradeoffs may not be relevant in the future, if the languages are no longer used or other languages are developed, combining the best features of the current languages. However, given the current state of programming languages, the tradeoffs are important today. The current implementations of clients can be constructed in any language, but unlike a client written in C or C++, an unsigned Java applet provides assurance that the client is not collecting personal data from the computer or using it for some other malicious purpose, such as acting as a spam relay or contributing to a distributed denial of service attack. Because Java is an interpreted language, it is slower than compiled languages such as C and C++, which means that Java clients are likely less productive than C or C++ clients. However, it is unlikely that volunteer computing projects will port their clients from C/C++ to Java, so they will likely choose to implement their clients only in one language. If they implement the client in C/C++, they will lose some potential volunteers who require the added security provided by Java. If they implement the client in Java, they will lose some of the effectiveness of the donated CPU cycles.

3.4 Summary

In this section, we discussed the two different categories of methods to improve volunteer computing project completion times, as well as the tradeoffs of different types and implementations of volunteer computing clients. The category of social methods focused on ways to increase the number of participants in volunteer computing projects. The technical methods include porting volunteer computing clients to devices other than general-purpose computers. We tested a client on the only video game console system that provides a user the ability to do this and found that it is a viable option but significantly less effective than a general-purpose computer with a CPU that has an equivalent clock speed. We also explained the findings of our past work where we tested different task retrieval methods. We have explained how clients can be implemented to increase security and simplify installation and discussed the different types of volunteer computing clients that can be created.

Although the social category of methods to increase the number of participants in volunteer computing projects is an important goal, this work focuses on the other category of technical methods to improve volunteer computing project completion times. We discussed the potential ways to improve the server portion of volunteer computing programs and found that there is very little that can be done to the server components to increase the effectiveness of volunteer computing projects. We discussed four client-based technical methods to increasing the effectiveness of volunteer computing projects: task retrieval policies, security, ease of installation, and providing multiple types of clients. Our past work shows that different task retrieval policies have a significant impact on the effectiveness of volunteer computing projects, but more work needs to be done to find better policies. We have explained how clients can be implemented to increase security and simplify installation and there is no need to explore this further. We have explained how providing different types of clients may increase the effectiveness of volunteer computing projects, but this needs to be explored to determine the productivity differences between the different types of clients. We perform a preliminary study of the productivity differences in Chapter 8.

4 Data Collection

In this chapter, we discuss the computer usage data we needed to collect to drive our simulations, the data collection process, some analysis of the data, and compare the data we collected with a similar data set. The information in this chapter comes from research we published in [82] with some minor modifications and additions.

4.1 Required Data

In order to perform accurate simulations and correctly explain what the best task retrieval methods are for various circumstances, as well as to correctly characterize the amount of work different types of clients can perform, we needed the parameters of the simulations to reflect those present in the real world. This required getting data from computer usage studies or existing volunteer computing projects. We determined that the data we needed had several requirements listed below.

Requirement 1: The data needed to accurately reflect the computers that might be available for volunteer computing.

Four major classes of computers that might be used for volunteer computing are:

1. Home computers: These computers are used for personal and family related activities.
2. Business computers: These computers are used for business related activities.

3. Public computers: These computers are available for the general public in some community to use, such as computers in a public library or computers in a lab on a college campus that is open to the entire student body.
4. Undergraduate student computers: These computers are owned by students and used at their universities or colleges.

We elected to use computers from the four categories we have described as we wanted some diversity in our samples. We reasoned that the usage patterns of those categories might be significantly different and thus we needed to collect data from computers in each category to see the entire picture.

Requirement 2: The number of computers we collected data from in each category listed in requirement one needed to be significant enough to make some reasonable observations about the data. Thus, we attempted to collect data from as many computers from each category as we could.

Requirement 3: The data we needed had to reveal when computers were available to participate in volunteer computing, unavailable to participate but powered on, and when the computers were unavailable to participate.

The periods when the computers were available and unavailable would be used for simulations involving the model of volunteer computing clients that run when some conditions are met, such as screensavers. The periods when computers were available and unavailable but powered on would be used for simulations involving the model of volunteer computing clients that run all the time. In order to collect this data, we needed to define when a computer was available for volunteer computing. We note here that unavailable is simply the complement of available.

There are several ways to determine when a system is available. Although some of the methods have shortcomings, we list them anyway for completeness. One way is to track when the user has last given some sort of input to the computer with the keyboard or mouse and assuming if the user has not given input to the computer in some amount of time, then the computer is available. This information can be obtained by querying the operating or asking users to provide the data. Another method is to track the computer's CPU utilization rate and assume the computer is available when the utilization drops below some threshold. However, this fails to take into account the fact that some screensavers are very CPU intensive. To fix this shortcoming, we decided that we could also test to see if the screensaver was running. However, some utilities such as antivirus or anti-spyware utilities may be running when the screensaver is also running and might use enough of the computer's CPU time to indicate that the computer was not idle. It was unclear whether it should be considered idle, however, if the computer was updating its virus definition files or running a sweep to detect spyware. Eventually, we decided that we should just do what volunteer computing programs do: we should consider a computer available if the screensaver is running.

Requirement 4: The data needed to represent the available, unavailable but powered on, and unavailable periods in a fine enough granularity to make our simulations accurate.

Because we decided to use the screensaver to define when computers were available, the data needed to have a fine enough granularity to reflect the state of the screensaver on computers accurately. Screensavers can be set to come on after a period of idle time that is a multiple of one minute. Sampling the state of the screensaver every 60 seconds would allow the data to be off by as much as 59 seconds, which we deemed was too inaccurate.

If we would need to collect the data instead of just using data from others, we wanted to ensure that a program we would design to collect the data would not impose enough of performance cost on the computer such that the user would notice the program was running. We conducted an experiment to help us decide on a sampling rate, running a prototype data collection program on an old computer with an Intel Celeron 900 MHZ CPU and 128 MB of RAM that was running Windows XP. We expected that if we had to collect the data ourselves, then any computer used in our study would have equivalent or better hardware. Thus, if our program did not have a noticeable performance impact on this computer, then we believed it would not have a noticeable impact on any computer that would participate in our study. The prototype was set to sleep for 60 seconds between every time it queried the operating system to see if the screensaver was running. By monitoring the CPU usage of the program with Task Manager for several minutes, we saw that the program was using 0% of the CPU. We decreased the sleep time to 30 seconds, and upon observing the program was continuing to use 0% of the CPU, we decreased it again to 10 seconds. With this setting, the program continued to consume 0% of the CPU. This setting allowed us to determine when the screensaver starts and stops within 10 seconds of the events actually occurring. This means our measurements would be accurate to the nearest minute, which we felt would be precise enough for our simulations.

Requirement 5: The sampling of the data needed to have been continuous for enough time to get an accurate representation of computer usage patterns and try to avoid the effects of anomalous data.

Although a year's worth of continuous sampling would provide us with more accurate results than shorter periods, we did not expect to be able to find continuous traces spanning even multiple months in a study and decided that at least two weeks of data would be necessary to ensure that outlying usage patterns did not skew the data we collected.

Requirement 6: The data we used for the simulations needed to have been recorded in a consistent manner.

We would only be able to use data from two or more different studies that together had collected data in the four categories of computers listed in Requirement 1 if both studies had recorded the same data using the same method and the same sampling intervals.

Requirement 7: The data needed to have been collected relatively recently.

Computer usage has changed significantly during the last 10 years, as computers have gone from being something only a small segment of the population could afford to being a commodity that a huge segment of the population can afford.

We contacted existing volunteer computing projects to see if they had collected this data, but both GIMPS and the BOINC-based projects do not collect this information [83, 84]. None of the other projects we contacted responded. Thus, we decided to review other studies relating to computer usage statistics in an attempt to get the necessary information. Although there were quite a few studies that collected data about the availability of computers, we were unable to find any that had data that was close enough to meeting our requirements. However, for completeness, we discuss the most relevant studies we examined.

4.2 Related Studies

There have been quite a few studies about the availability of computers. Wolski et al. gathered data and analyzed it along with data from another study in an attempt to predict the availability of desktop computers [85]. One of the data sets they gathered was from machines in a Condor pool at the University of Wisconsin [85]. This data set measured the time that a process was able to run on a computer because the Condor framework determined that the workstation was idle [85]. This method only measured the idle time of some computers at some times, thus collecting only a fraction of the available data points, as the method for gathering the measurements was to submit 10 jobs that simply measured the idle time to the Condor pool [85]. The other data set they gathered was from a set of computers in several labs at UCSB that are accessible to computer science students [85]. This study measured the time between reboots of the computers to determine the availability of the computers [85]. We note that this data does not represent the availability of computers for volunteer computing, but rather the time the computers were powered on. The final data set that they examined was one collected in 1995 by Long, Muir, and Golding that collected data about 1170 computers connected to the Internet that responded to randomized rpc-statd calls [85]. Although Wolski et al collected data from several different sources which is very important to our work, the way they measured availability was not consistent across the different sources and was not consistent with our method of defining availability. Thus, we were unable to use their data for our work.

Mutka and Livny collected data from three different types of users [86]. They monitored computers used by graduate students, faculty, and systems programmers [86]. Mutka and Livny considered workstations to be unavailable when they are used or when the average user CPU usage was greater than one quarter of one percent within five minutes of being used by the owner [86]. However, the data is almost 20 years old and more importantly, only 11 computers were monitored for their study [86]. Because of this, we did not feel that the data was representative of the data we needed.

Acharya et al examined traces of three different sets of workstations [87]. For the trace from the University of Maryland Computer Science department’s cluster of public computers, a computer was considered available if the CPU utilization stayed below 0.3 for five minutes [87]. In the trace from a Condor pool of roughly 300 workstations at the University of Wisconsin, a computer was considered to be available when the Condor software deemed it so [87]. The remaining trace came from a group of computers at UC Berkeley [87]. Again, the inconsistency of determining when computers are idle and the data only coming from one type of computer rendered this data unusable for our work.

Kondo et al published results of a study that provided a data set that most closely resembles the data that we needed for our simulations [88]. This study, intended to be used for the study of enterprise desktop grids, measured the availability of over 200 computers in the San Diego Supercomputer Center (SDSC) [88]. Kondo et al recorded whether each computer was powered on and reachable over the network and the percentage of the CPU time that was available for a distributed application at 10 second intervals [88]. Their recorded data did distinguish between a host being available, unavailable and powered on, and unavailable and powered on in a way that we might have been able to use [88]. In his dissertation, Kondo used a data set gathered in the same manner from a set of student lab machines [89]. However, because their measurements came from only two types of computers (what we deem business computers and public access computers, as opposed to a student, and home computers), we still needed to collect data from the other two types of computers [88, 89]. In order to keep the data we collected completely consistent with one measuring scheme, we chose not to use the business computer data they had collected. However, in Section 4.5, we compare the results of Kondo's study with our results. It should be pointed out that Kondo's method of data collection differs from ours significantly. While his method collected data for 28 days, the 28 days were split into four sets of date ranges [90]. In contrast to this, our data was collected over 28 consecutive days. The difference between these data collection methods was another reason we were unable to use Kondo's data for our simulations.

4.3 How We Obtained Computer Usage Data

Because none of the studies in Section 4.2 collected data that met our requirements and we were unable to find any other sources with the data we need for our simulations, we needed to collect the data ourselves. It appeared that the most accurate way to collect the information would be to have a program collect the data automatically from users, rather than having to rely on the accuracy of users' memories and their honesty. Our method also puts the burden of collecting the information on us and allows users to expend a minimal amount of time and effort to get us the information we want. We believed this would likely yield us more information than surveying users.

4.3.1 Development Decisions

We needed to make decisions about several aspects of such a program that would collect this information. We needed to determine the operating system(s) for which we would develop the program, the type of program we would use to collect the information, the method of recording the data, the method of collecting the data, and the duration of the experiment.

4.3.1.1 Target Operating Systems

We took two factors into account when determining the operating system(s) on which our program would run. The first requirement we needed to satisfy was to be able to collect large quantities of data with minimal effort. This meant that a good strategy would be to write the program for a single operating system if possible. Thus, if we could determine an operating system with a large enough pool of users, writing the program for that operating system would be preferable to writing the program and having to port it to several operating systems. Our second requirement was to be able to collect data from several of the following different categories of computers: home computers, college student computers, business workstations, and public access workstations such as computers available for use by members of college campuses like library or computer lab workstations. We wanted to collect data about each of these types of computers because we felt that the usage patterns would be significantly different on each type and that these types of computers (as opposed to servers or some other types of computers) would be the most likely to participate in public resource computing projects. The family of Microsoft Windows operating systems was the only operating system that satisfied both of our requirements. This family of operating systems is used on all of the types of computers about which we want to gather information, including the public access computers on the Worcester Polytechnic Institute (WPI) campus (and other campuses we are aware of). We are unaware of another operating system that meets these requirements. The family of Windows operating systems is also used by many people and we believe is the predominant operating system in use for these types of computers, so we believed we would have a large enough pool of users.

4.3.1.2 Type of Program

Having settled on the Microsoft Windows family of operating systems as our target platform, we had three types of data collection programs that we could write: a Windows Service, a Windows application, or a screensaver. There are benefits and drawbacks to each of the three types of programs. The benefits of a screensaver are that they provide an easy way to inform the user of anything they need to know and also provide a way to entertain the user and educate them about the program. Far more importantly, screensavers receive messages from the operating system to start and stop. Thus, one could create a screensaver that upon receiving those messages could log whether a user is idle. This ability makes a screensaver seem like the logical choice at first. However, after talking with several users, it became clear that some people like to run particular screensavers and thus asking users to run one of our own design would put too much of a burden on some users and decrease our pool of potential users. In addition to this, for variants of Windows, each user may set his own screensaver, so all users of a computer would need to use our screensaver to make the data collected valid. Another problem with this approach is that a different screensaver can run when no user is logged onto a computer, so the default screensaver would also have to be set to our screensaver to make this method work. Since screensavers are also easily changed, users could easily accidentally or maliciously change the screensaver during our experiment and invalidate the results. The combination of these deficiencies makes the screensaver an unattractive option.

Writing our program as a Windows application provides few benefits and suffers from many drawbacks similar to those of screensavers. A Windows application provides a way to communicate with the user to both inform the user about various issues as well as a way to get user input. However, a Windows application does not receive the same messages as a screensaver does to start and stop unless it is the foreground application. It is unrealistic to assume that the application will always be the foreground application when the user becomes idle, so this would not work without some additional code. The application could use operating system hooks to register to receive all keyboard and mouse events, do any necessary processing based on the events, and pass them on to their intended destination application. Sidney Chong wrote a dynamic link library (dll) for the Windows operating system to determine when a computer is idle [91,92]. The dll installs hooks into the operating system so that keyboard and mouse events are detected and the times when they occur can be recorded [91,92]. However, using the hooks clearly incurs overhead. In addition to this, the code does not always work correctly, and after about 10 minutes, the callback function sometimes stops working correctly [93]. We verified that this is indeed true. In addition to these shortcomings, users would be able to see the application in the "Applications" tab of the Windows Task Manager and the users might end then task accidentally or maliciously, which would invalidate the data collected. The application would also need to be in each user's startup folder on all computers, which would be difficult to accomplish on public access workstations. In addition to this, the application will not run when no user is logged in, which will makes the results inaccurate.

Fortunately, Windows services can overcome most of the shortcomings of both screensavers and applications, while suffering from few enough of their own shortcomings to make them a viable type of program to use to collect the data. A windows service can easily be set to run all the time a computer is powered on, regardless of whether a user is logged onto the machine or which user is logged on. The service can also be set to start whenever the computer is powered on. In addition to this, a service does not show up in the "Applications" tab of the Windows Task Manager. Thus while services can be stopped by a user, it is much harder to for a user to stop a service accidentally or maliciously, because it is difficult for a user to know what the service he is stopping does and thus users appear reluctant to stop random services. Services do not have a method of communicating with the user the way an application or screensaver does, other than through log files. However, we decided that we did not need to communicate with the user, so this is not a problem. The major drawback of a Windows Service is that it can only be run on Windows NT, Windows 2000, and Windows XP. However, Windows XP is far and away the dominant version of Windows among our target user pool, as only a couple versions of Windows besides Windows XP are still supported by Microsoft. The benefits of a Windows service and its minor drawbacks made it the logical choice for the type of program to use.

4.3.1.3 Method of Data Recording

In order to not lose any accuracy of the measurements that our program takes, the program needed some way to record the data in persistent storage so that if the computer was powered off by any means, the data would not be lost. However, just recording the status of the screensaver and the time when the measurement was taken every 10 seconds could easily use one MB of disk space per day. While this is not a problem for modern computers, we wanted to minimize the data sent back to out data collection server to try to avoid using a lot of the CPU time on the computers participating in the experiment. We also wanted to try to avoid any problems caused by the potential congestion of hundreds of computers trying to contact the server in a short time period. Our solution to this problem was to write the time when a measurement was taken to the file only if the state of the screensaver changed from off or an unknown state (which only occurs when the service is started or restarted) to on. If the screensaver was running at the previous point in time when its state was sampled and it is still on at the current time, then the character "*" was written to the file. If the screensaver was not running at the previous point in time when its state was sampled and it is still not on at the current time, then the character "@" was written to the file. This would allow us to parse the data file after it has been collected and not lose any information, while compressing the size required to store the data file to significantly less than one MB. In fact, in a worst-case scenario barring any strange anomalies, we noted that the file would be less than 27 KB.

The data file generated will be the biggest when the most number of times are written to the file, since they are clearly more characters than a single character. We noted that Windows sets the minimum delay for the screen saver to one minute. Thus, the screensaver may change state from being off to being on at most every seven intervals. There are 6*60*24 = 8640 intervals during one day of the experiment, so the time can be written at most 1235 times in one day. This assumes that the user is not constantly previewing the screensaver, which could cause the switch to occur every other interval. At a cost of 1235 * 16 = 19760 characters plus an additional 8640 – 1235 = 7405 characters, the most characters that would be written to the file is 19760 + 7405 =

27165, which is equal to a little less than 27 KB. Thus, the maximum size of the file is about 1/40th of the size of what it might otherwise be.

4.3.1.4 Method of Collecting the Data

We chose to have the computers the experiment was running on send the data collected to our data collection server every 24 hours to minimize the data lost if a person accidentally or maliciously sabotaged the experiment on a computer. While this was not necessarily the best way to minimize the result of sabotage, we decided that it would be adequate, better ways would require significantly more effort and we were not overly concerned about sabotage.

4.3.1.5 Experiment Duration

The experiment needed to produce enough data in order to prevent anomalous behaviors from making the data collected inaccurate. The experiment also needed to be short enough to generate results quickly enough so they could be analyzed and put to good use. In addition to this, because we planned on collecting data from computers used by WPI undergraduate students, we needed to make the experiment run during a seven-week period so the collection did not coincide with the undergraduate vacation. We believed that computer usage would not be similar to that during a non-vacation time. We chose to run the experiment for four weeks to make it short enough to meet all these requirements.

4.3.2 Service Design

The service needed to be designed to be as robust and non-intrusive as possible. As explained in Section 4.3.1.4, we wanted the data from each computer to be sent to the server every 24 hours. However, we wanted the data collection to continue during the process of sending the data to there server, so the service has a thread that collects the data and stores it in a different file for each day. A second thread sent the completed data files to the server, and a third thread kept the sending thread informed of which data files are ready to send. This method allowed the service to store all the data to the disk, so the service could recover without any problems if it was stopped and restarted. The source code for the data collection service is in the data_collection.zip file that is available where you downloaded this dissertation.

4.4 Results and Analysis

We ran a prototype version of our service on a number of WPI computers in the CCC labs and on some student computers during the Fall of 2005. This pilot program proved that we could collect the data we need. The final version of the program ran on public access workstations on the WPI campus, computers belonging to WPI undergraduate students, home computers, and computers belonging to a business which participated on the condition of anonymity. The program ran on the WPI public lab computers between January 26, 2006 and February 28, 2006. The program was run on student computers between March 13 and April 25 of 2006 and on home computers between March and April of 2006. The program was run on the business computers between July and August of 2006. We obtained traces of 68 public computers from our university which were split between 3 computer labs, 38 undergraduate computers, and 25 home computers, and 26 business computers.

During the data collection process, we experienced a server failure which destroyed some of the data files that were sent to us by student computers. Therefore, we had to adjust the traces to account for the lost data files. In order to do this, we manually spliced the traces together from the last complete interval before the missing data file to the first complete interval after the missing data file. This means that some of the traces were shortened from 28 days. Once we had assembled all the traces from the computers, we converted each trace into a separate file that showed the length of the trace (which did not include any missing time) and the periods when the computer was available for volunteer computing, unavailable for volunteer computing but powered on, and when the computer was powered off. These files are available in the traces.zip file that is available where you downloaded this dissertation.

Once we had completely processed and assembled the traces, we examined them to determine if the traces themselves contained any useful information about computer usage patterns beyond just the data we would use to drive our simulations. We found that the average of the total percentage of time during the studies that different types of computers were available for volunteer computing varied greatly between the different types of computers, ranging from 73% for public computers to 27% for home computers and business computers to 18% for student computers.

We observed that the amount of time computers were available for volunteer computing varied greatly by computer within the undergraduate student, home user, and business types of computers with a range of over 80%. However, the amount of time the public computers were available for volunteer computing varied significantly less by computer, as shown in Figure 4.

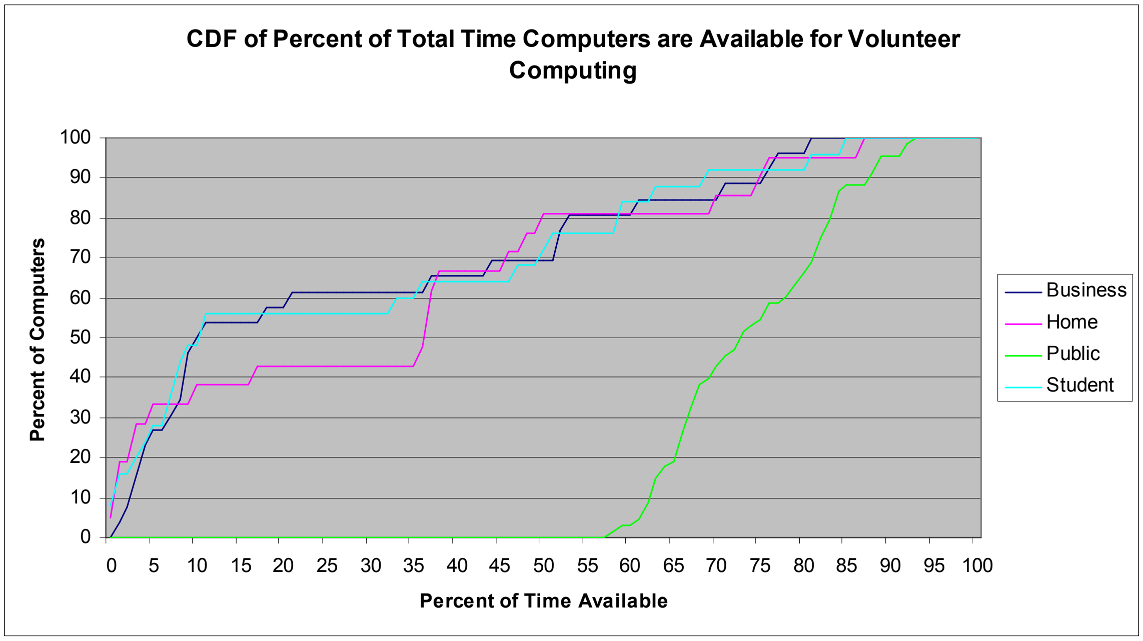


Figure 4 - Computer Availability

The average durations for the available, unavailable but powered on, and unavailable periods differed significantly between the different classes of computers, as shown in Figure 5.

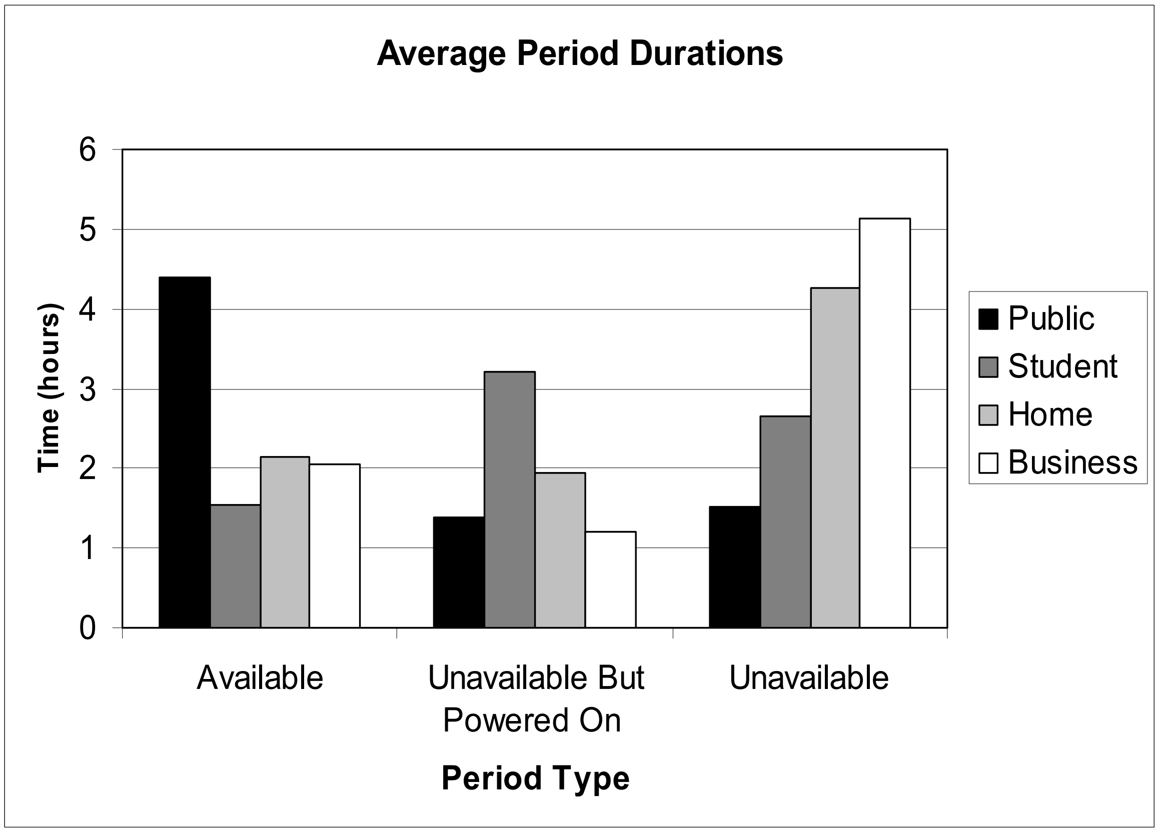


Figure 5 - Average Period Durations

The cumulative distribution functions for the available periods differed a bit as shown in Figure 6, although we found that the cumulative distribution functions of the other periods were relatively similar for the different classes, as illustrated by Figure 7 and Figure 8.

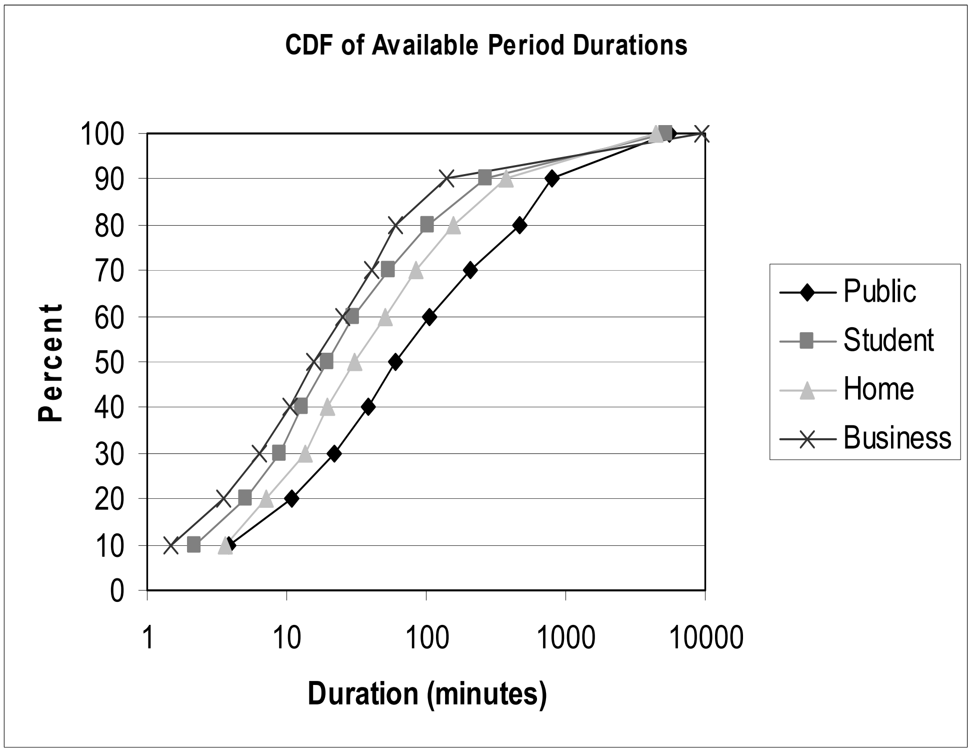


Figure 6 - CDF of Available Period Durations

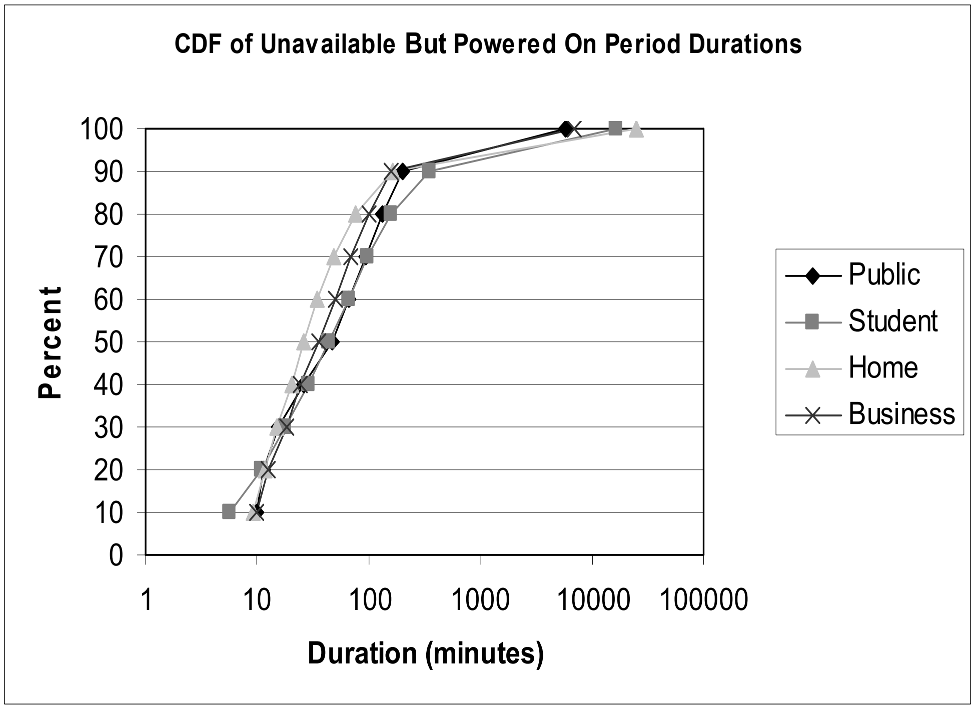


Figure 7 - CDF of Unavailable but Powered On Period Durations

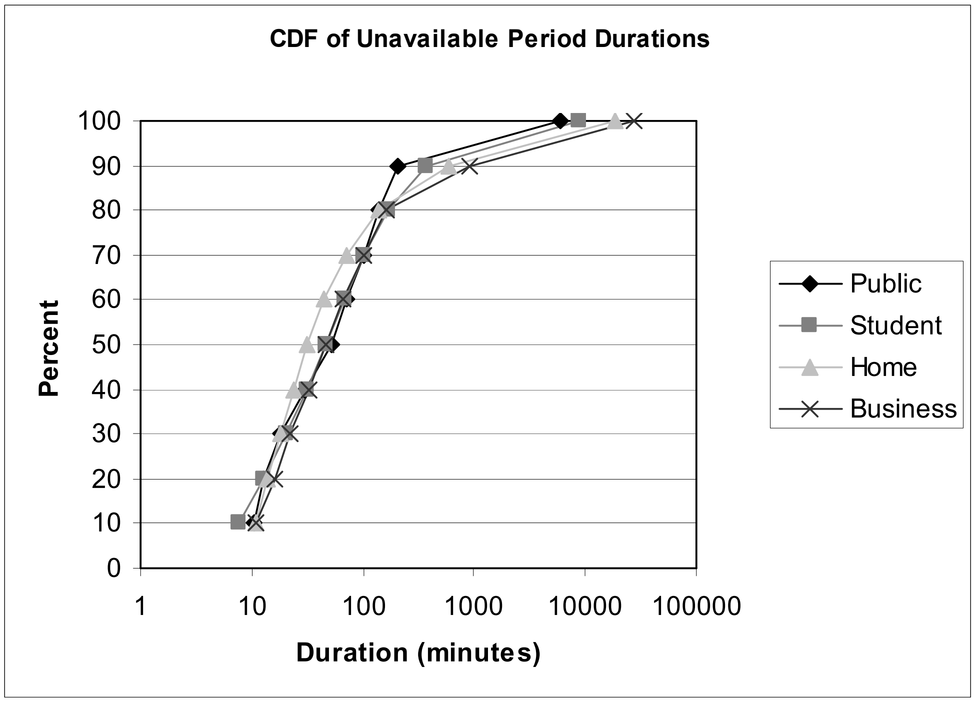


Figure 8 - CDF of Unavailable Period Durations

Although the CDFs of the different durations appear similar between the types of computers, closer inspection of the data on a per computer basis shows that for the student, home user, and business classes of computers, some computers in the same class that exhibit very different usage patterns. In particular, we found there were computers that were only powered only for short periods of time throughout the data collection period. These computers were almost never available for volunteer computing. However, there were some computers that were powered on for almost the entire duration of the data collection period. Many of these computers were available for volunteer computing for a significant portion of the day.

Using the Arena program from Rockwell Automation [94], we calculated the most likely distribution for the available periods for each computer. We found that the best match for almost all of the lab computers from two labs was a Weibull distribution.

However, despite this being the best match, Arena indicated it was not a good match at all, as the p-values from the Kolmogorov-Smirnov Goodness-of-Fit Test were very large. The best match for almost all of the computers in the third lab was a Beta distribution. The p-values were much better for this lab, but although that does not indicate the Beta distribution is a bad characterization, it also does not show conclusively that it is an accurate characterization. The best matches for the available periods from the home, student, and business computers were very inconsistent, being split between five, six, and four different distributions respectively. In addition to this, once again many of the p-values Arena produced for the Kolmogorov-Smirnov test indicated that the distributions were generally not good matches for the data.

In addition to trying to determine if the available periods matched any well-known distributions, we also attempted to determine if there were groups of clients that exhibited similar behavior such as very frequent availability for volunteer computing or very infrequent availability for volunteer computing. We felt that if such groups existed, there might be value in developing different task retrieval policies for the different groups. Because we were looking to break the data into groups all at once, we used partitional clustering. We used the k-means method of clustering in the SPSS software package. In k-means clustering, one specifies the number of clusters and the software partitions the data into that number of clusters. While there are other methods of partitional clustering such as fuzzy clustering, they were not conducive to splitting the data into groups that would be useful for us. We found that our efforts to classify the computers into different groups based on their availability for volunteer computing did not yield any significant results.

4.5 Comparison with Kondo's Results

As discussed in Section 4.2, there were some differences between the data collection method used by Kondo et al and the data collection method we used. However, because it is the most similar data to what we collected, we compare some of the results here. Kondo characterized the availability of the computers in terms of availability and unavailability, breaking both categories down by business hours and non-business hours. Kondo's work focused on enterprise desktop computing which is essentially a form of volunteer computing where all the volunteered computers come from within a business and is done over the business's network and the main task is one the business is trying to solve. Because of this, it was important for him to break down his data into business hours and non-business hours. He might use different heuristics of assigning tasks during business hours than in non-business hours. Our data is not broken out by business hours vs. non-business hours. In our data, we do not have all of our data coming from business computer the way he does. Therefore, business hours versus non-business hours is not a useful metric for us, especially because public, undergraduate student, and home computers do not necessarily have the regular usage patterns business computers are likely to have.

Kondo found that for his San Diego Super Computer Center (SDSC) traces, during business hours, 70% of the available intervals were 2 hours or less in duration and the mean time was 2.0372 hours [90]. During non-business hours, available periods that lasted only 2 hours or less only accounted for about 36% of the intervals and the average interval length was 10.2327 hours [90]. Our most comparable data set to Kondo's SDSC traces are our business traces. We found that the 70% of available periods for the business computers were 41 minutes or less in duration, far less than Kondo's results. 70% of our home computers available period durations were 85 minutes or less and 70% of the student computers available period durations were 54 minutes or less. Our public computers did have longer available period intervals in general, with 70% of them being approximately 3.5 hours or less in duration.

We were able to compare the percentage of time that computers were available for the computers from Kondo's SDSC traces and from our traces. We felt that the business and public computers from which we collected data, rather than the home computers and student-owned computers, were the computers that most closely resembled the SDSC computers. We downloaded Kondo's SDSC traces which are available at [95] and wrote a program to process the SDSC traces and calculate the percentage of time that each of the SDSC computers were available. Using that information, we generated the CDF of the percent of total time that computers were available for volunteer computing, shown in Figure 9.

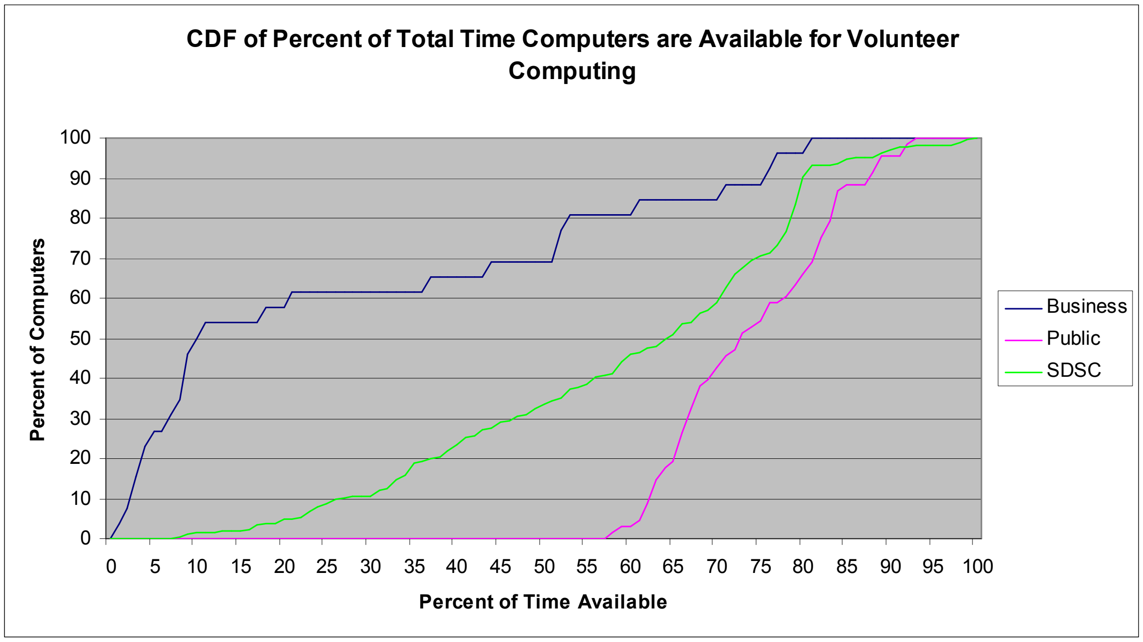


Figure 9 - Comparison of Kondo's SDSC Traces and Our Business and Public Computer Traces

The CDF of availability shows that the distribution of the percent of time that the SDSC computers were available was different than the distribution of the percent of time that the business and public computers from our study were available. We do note that there are a couple of differences in how the data from the SDSC computers was collected and how the data from the computers in our study was collected which may have affected the distributions. While our data was collected continuously for 28 days, the SDSC traces span 9 days that are not all contiguous. Thus, while our traces did include weekends, the SDSC traces we downloaded did not. Our traces also spanned 24 hours per day, while the SDSC traces only spanned eight hours per day. Thus, the SDSC data set and our data set are clearly useful for different purposes.

4.6 Summary

In this chapter, we discussed the data we needed for our simulations. We discussed the related work and why the data collected in related studies was not adequate for our simulations. We explained our methodology for collecting the traces of computers that we would use in our simulations, why we designed our data collection program as a Windows service, and how it worked at a high level. We also discussed the results and our attempts to analyze the data to see if it matched any well-known distribution and if we could break it into useful clusters. The data did not match any well-known distribution. Our attempts to cluster the data did not provide us with any additional information. We compared our traces to Kondo's SDSC data sets and found that the distributions of percent of time the computers were available was different.

5 Simulations to Examine the Effects of Task Retrieval Policies

In this chapter, we explain how we used simulations to explore how using different task retrieval policies affects the amount of tasks that volunteer computing clients complete, based on the data we have collected about computer usage. In order to do this, we developed a discrete event simulator that would use the traces we collected as input to a trace-based simulation. The information in this chapter and Chapter 6 was published in [96].

5.1 The Simulator Design

This simulator improved upon our work from Section 3.2.3.1.2 in three ways. First, the simulator used actual data collected from computers. Second, the simulator incorporated checkpointing, making the results more accurate. Third, we replaced the abort multiple parameter with a parameter called the delay bound. The delay bound serves the same purpose in the new simulator as the abort multiple did in the simulator used for our work in Section 3.2.3.1.2 and is discussed in more detail in Section 5.1.1. However, the delay bound is the standard name used in volunteer computing to describe the parameter. The parameters used by our simulator are similar to those used by simulator used for our work in Section 3.2.3.1.2. The only differences between the two sets of parameters is that the new simulator uses the number of tasks buffered in place of the number of hours buffered used by the old simulator, the simulator uses several new task retrieval methods, and the abort multiple has been replaced with the delay bound.

5.1.1 Simulation Parameters

The parameters for the new simulator were:

- Task retrieval method. The method clients use to retrieve tasks.
- Number of tasks buffered. The number of tasks a BOINC client buffers (not relevant for the *Buffer1*and *Buffer None* methods).
- Delay bound. This is the limit on the amount of wall clock time that may pass between when a task is downloaded and when that task is aborted if it has not been completed.
- Download speed. The client's download speed (in bps).
- Completion Time. The time it takes to complete a task (in hours) if it is not interrupted. It is assumed that all tasks take the same amount of time, specified by this parameter, to complete.
- File size. The size of the file required by each task, that needed to be downloaded before starting a task (in MB).

5.1.2 Assumptions

We make similar assumptions here to the assumptions that we made in Section 3.2.3.1.1. These assumptions are that files are small (10 MB or less), download speeds are fast (300 kbps or faster), tasks take a long time to complete (two hours of CPU time or more), and the computers are constantly connected to the Internet. Based on these assumptions, we note that downloading a task should always take significantly less time to complete than performing a task and therefore, a client that buffers tasks or initiates the download of a task while it still has some non-trivial amount of a task left to complete will always have a task available to process. This agrees with what we observed in practice. However, we do acknowledge that if downloading a task takes longer than it takes for the client to complete a task, then it would be possible that a task might not be waiting in the buffer and there would be wasted time waiting for the download to complete. We also assume that the parallelism of a modern computer allows the computer to do other work while downloading a file. If this was not the case, then a computer would freeze and become unresponsive whenever on downloaded a file. We assume that the fraction of CPU time required to accomplish the downloading of a file is very small in comparison to the total amount of wall clock time that downloading a file takes and thus that downloading a file does not take many CPU cycles away from a workunit currently being processed.

5.1.3 Task Retrieval Policies

Our preliminary work in Section 3.2.3.1 showed that using different task retrieval policies may result in significantly different amounts of donated CPU time being wasted. Therefore, we believed that one might be able to create a policy that resulted in more tasks being completed than the existing policies. We examined the existing policies, created additional policies using the best features of the existing policies, and evaluated both the existing and new policies. The existing policies used by current volunteer computing projects are a policy where no tasks are buffered and a policy where some number of tasks are buffered. We refer to the policy that does not buffer any tasks as *Buffer None*. We have revised the policy that buffers multiple tasks from our preliminary work to behave the way the task retrieval policy is implemented in BOINC-based projects, making it more realistic. We refer to the revised version as *Buffer N Days*. Our preliminary work showed a significant amount of donated CPU time may be wasted when tasks are buffered and that the amount of wasted time increased as the number of buffered tasks increased. Despite the wasted time associated with buffering, a benefit of buffering tasks is that no CPU time is wasted as a client downloads the next task. However, this benefit can be gained by buffering only one task, rather than buffering multiple tasks, and since previous work indicated that buffering more tasks resulted in more wasted CPU time, a policy that buffered one task might provide an improvement over a policy that buffered no tasks and a policy that buffered multiple tasks. Therefore, we chose to test a policy that buffered one task, which we refer to as *Buffer One Task*. We point out that the time a client finishes downloading a task is an important aspect of task retrieval policies because the client has a fixed amount of wall clock time to complete a task once the client finishes downloading the task. If the task is not completed, all the time spent working on it is wasted. We decided that another way to gain the benefit of having a task buffered while increasing the likelihood of a task being completed was to have the download of a task finish just before the task a client was working on is completed. This led us to develop the *Download Early* policy which starts downloading the next task when the current task is 95% complete. In order to fully understand how well the policies we test perform, we need to understand how well a policy could perform theoretically in the best case. This best case policy, which we refer to as *Optimal*, would allow us to determine how much improvement might be attained by comparing its results with the results from other policies. The comparison would demonstrate whether it made sense to devise additional policies other than the ones we tested, in order to gain a further improvement.

The task retrieval methods that the clients use are

- ***Optimal:*** The *Optimal* policy produces an upper bound on the number of tasks a client could complete. Therefore, it has complete information about the availability of the computer and it assumes that downloading a task takes no time at all. Using the computer's availability information, the client determines if it can complete a task if it were to download a task. If the client could complete a task, it downloads it. Otherwise, the client sleeps for a second and re-evaluates whether it could complete a task if it downloaded one. We point out that this policy cannot actually be implemented because it requires precise knowledge about a computer's future availability and assumes that it takes no time to download a file. Thus, the results obtained by using this policy cannot actually be achieved.
- ***Buffer None:*** The client does not buffer any tasks and retrieves the next task after it has completed or aborted its current task.
- ***Download Early:*** When the client has completed some percentage of its current task (we used 95 %), it downloads the next task.
- ***Buffer One Task:*** The client keeps one task in its buffer. Upon completing a task, the client begins the one in its buffer and simultaneously downloads a new task to replace the one that it has just removed from the buffer.

- ***Buffer N Days:*** The client keeps N day's worth of tasks in its buffer, refilling the buffer when it has only one task remaining in it.

5.1.4 How the Simulator Works

The simulator partitions time into one-second intervals. In the data we collected, one second is the largest unit of which the durations of the periods are all multiples. Because of the way we collected the data, this is the finest resolution of time that we can use. Therefore, using one-second intervals gives us the most accurate simulation results. Because the time units are each only one second, we allocate the full time unit to one task that is performed during each interval. Therefore, although a task might only require a fraction of a second to complete, we charge a full second for that task. We note that trying to get finer resolution is unnecessary and would require us to have a very accurate understanding of the hardware devices and how they interact with each other.

We implement checkpointing by assuming that it takes a fixed amount of time to create a checkpoint and the same amount of time to restore the state of a task from a checkpoint. The time to create a checkpoint and to restore the state of a task from a checkpoint is a parameter in our simulations and is discussed in detail in Section 6.2, along with how we calculated the optimal time in between checkpoints. When the optimal amount of time between checkpoints has elapsed, the client creates a checkpoint. We also assume that checkpointing and restoring the state of a task from a checkpoint can be interrupted. If they are interrupted and the computer is turned off before the operation has been completed, then the operation is deemed to have failed. Similarly, the act of downloading a workunit is deemed to have failed if the download is not completed before the computer is powered off. The source code for the simulator is in the simulator_source_code.zip file that is available where you downloaded this dissertation.

5.2 *Testing the Simulator*

We ran a set of tests on the simulator to ensure that it would work properly. We constructed mock traces of a computer's usage that were nearly identical to the traces we collected in Chapter 4 except that the mock traces covered shorter periods of time than the 28-day traces we collected. For each mock trace, we determined how many tasks a computer would complete if its usage patterns corresponded to the mock trace. We determined the number of tasks that a computer using a mock trace would complete by determining what each second in the trace would be used for. Then we ran the traces through the simulator to confirm that the simulator produced the output we expected it would. We designed tests to ensure that the downloading of workunits functioned correctly for all of the various download policies we implemented. The tests were also designed to ensure that the simulator performed checkpointing correctly and that workunits were completed and aborted at the correct times, based on the traces. The specific tests we ran were designed to:

1. Ensure that tasks were downloaded correctly when the simulator used a *Buffer None* policy and some tasks are completed on time while others are aborted.
2. Ensure that tasks were downloaded correctly when the simulator used a *Buffer 1* policy and some tasks are completed on time while others are aborted.

3. Ensure that tasks were downloaded correctly when the simulator used a *Buffer None and download when the current workunit was 95% complete* policy and some tasks are completed on time while others are aborted.
4. Ensure that tasks were downloaded correctly when the simulator used a *Buffer Multiple* policy and some tasks are completed on time while others are aborted.
5. Ensure that workunits checkpointed at the appropriate times and restored to the appropriate state after being restarted.
6. Ensure that workunits completed when they were supposed to according to the traces and ensure the workunits were aborted when they were supposed to be according to the traces.

The traces we used to test the simulator are given in the simulator_testing_files.zip file that is available where you downloaded this dissertation. We also ran some sanity tests to ensure that the simulator ran properly. The sanity tests were designed to show that adjusting the simulation parameters had the correct effect on the results of the simulation. We ran a simulation on a trace using each of the four original task retrieval methods. These simulations served as a control group. Then we doubled the completion time of tasks and re-ran the simulations to ensure that about half as many tasks would be completed. Finally, we doubled the delay bound to test that more tasks might be completed instead of being aborted and ran all four simulations again. We used the trace business_trace_10.trc for all the runs. The results of the sanity tests are shown in

Table 2. For the sanity tests, we used the checkpoint interval of 8640 sec and the checkpoint create and restore times of 10 sec.

Table 2 - Simulator Test Results

Test	Policy	File	Comments	Completed Workunits	Aborted Workunits
1	Buffer None	1-1	Control	20	0
		1-2	Completion Time = 2 X	10	0
		1-3	Delay Bound = 2 X	20	0
2	Buffer None Download Early	2-1	Control	20	0
		2-2	Completion Time = 2 X	10	0
		2-3	Delay Bound = 2 X	20	0
3	Buffer One	3-1	Control	20	0
		3-2	Completion Time = 2 X	10	0
		3-3	Delay Bound = 2 X	20	0
4	Buffer Multiple	4-1	Control	18	2
		4-2	Completion Time = 2 X	10	0
		4-3	Delay Bound = 2 X	20	0

5.3 Summary

In this chapter, we discussed the simulator we created to examine the effect that using different task retrieval policies had on the number of tasks that volunteer computing clients completed. We discussed the simulator and its features, our assumptions, the parameters it used, and the improvements over the simulator we used for previous work. We also explained the design of the simulator, as well as our methodology for verifying that the simulator worked correctly.

6 Simulation Results

We ran our simulations for many different parameter combinations. We first present the information we gathered in order to judge which parameter combinations are most likely to occur in volunteer computing projects. Then we discuss the method we used to obtain the values for how frequently the simulator should create checkpoints. Next we discuss the parameter values we used. Then we present the results of the simulations using the parameter combinations we feel are most likely to occur in volunteer computing projects. Next we describe the general trends we observed by varying parameters one at a time. Then we discuss the effects that some networking values would have on the results of our simulations. Finally, we present a summary of the overall results of all the parameter combinations.

6.1 Parameter Values from Existing Volunteer Computing Projects

We attempted to determine which parameters settings would be most likely to occur in actual volunteer computing projects. As the number of volunteer computing projects and the awareness of them have spiraled upwards since the beginning of 2005 when we began this work, significantly more information about the parameters used in projects has become available. In

Table 3, we present the information about parameter settings used in actual volunteer computing projects that we were able to obtain.

Table 3 - Current Volunteer Computing Project Parameter Settings

Project	File Size	Delay Bound	Task Duration	Checkpointing Frequency
SETI@home	350 k [97]	4 to 60 days [98]		10 seconds [99]
Folding@home	normal < 5 MB, some new large ones ~5 MB [100, 101]	max(10 days or 2 + 30*days to do on dedicated P4-2.8 machine) [102]	2-4 days on benchmark machine (P4-2.8) [103]	15 minutes [102]
Einstein@home	4.5 MB, 12 MB, 16 MB [104, 105]	1 week [105]		
QMC@home			4-48 hours [106]	
LHC@home		8 days, then 7 days, then 5 days [107, 108]		
Rosetta@home	800k, 1 MB, 1.2 MB, 1.5 MB, 3 MB [109]	(672 hours) changed to 168 hours [109]	3 hours, up to 24 hours, 2-4 days [109, 110]	varies based on user settings [110]
Grid.org	10k - 100k [109]	100 hours, 150 hours, 200 hours, 222 hours, 336 hours [109]	20 hours on PII-400 [109]	
Climateprediction.net		347 days 5 hours 20 minutes or 150 days 5 hours 20 minutes [98]	3 weeks, 8 weeks, 20 weeks [111]	15-30 minutes [99]
SIMAP	1-2 MB [112]	10 days [98]	2 hours [112]	
The Riesel Sieve Project		7 days [113]	30 minutes [114]	10 minutes [115]
World Community Grid			10-20 hours [116]	

We see that the file size varies 10 KB and 16 MB and seem to fall in groups of files that are less than 1 MB (small), files that are approximately 1 MB to 5 MB (medium), and files that are 12 MB to 16 MB (large). We attempted to select one file size to be representative of each group for our simulation parameters, and selected file sizes of 1 KB, 1 MB, and 10 MB. Because the bulk of the file sizes were in the 1 MB to 5 MB size range and most of those were closest to 1 MB, we chose the 1 MB file size as the base value for our simulations.

The bulk of the delay bounds are clustered around 7 days, with longer and shorter delay bounds. It was clear that 7 days should be the base value for delay bound for our simulations. We also chose a longer and shorter delay bound so we could determine the effect of varying the delay bound. We chose to double the 7 days as the higher value and to take half of the 7 days as the lower value to try to provide realistic approximations of how one might tune the delay bound for a volunteer computing project. Going lower than 3.5 days would require very consistent computer usage of the client and not allow more than a very small number of tasks to be buffered. Going higher than 14 days would begin to let tasks be turned in very late which might begin to impede the timely progress of a volunteer computing project.

The task durations varied from just 30 minutes to many weeks, with the bulk of the durations being between 2 and 48 hours. We selected task durations of 4 hours and 24 hours as our base values, and also ran our simulations with tasks of length 2 hours, 8 hours, 16 hours, and 48 hours to try to cover the parameter space better.

For the download speed of the clients, we chose to test 10 mbps and 300 kbps. We selected 300 kbps as the base value, as we believe that it approximates the home user bandwidth better than 10 mbps and home users appear to be more likely to participate in volunteer computing than businesses do because of security considerations.

6.2 Checkpointing

In addition to gathering information about the parameter settings from current volunteer computing projects, we also attempted to determine reasonable values for how long it would take a science application to create a checkpoint and how often checkpoints would be created. We reviewed work on optimal checkpoint creation from several authors, including the seminal work from Young [117], and Gelenbe & Derochette [118, 119]. We also reviewed the more recent work from Vaidya [120] and Plank & Thomason [121].

In order to determine the optimal checkpoint interval, all of the authors agreed that one needs to know the failure rate of the process being checkpointed. Using the system traces we collected, we calculated the failure rate. We found that the traces covered 356418410 seconds and 2878 failures occurred during that time, giving us a failure rate of one failure every 34 hours. Another critical value that all of the authors agreed upon is C, the time required to calculate the optimal checkpoint interval is the time it takes to create a checkpoint. According to Vaidya [120], if C is small in comparison to the mean time between failures, then it can be verified that Young's approximation is correct. Young's approximation is that the optimal time between checkpoints is $\sqrt{2 * C * mean\_time\_between\_failures}$ [117]. Although it is not necessarily consistent how long it will take to create a checkpoint, given that many volunteer computing workunits take on the order of a few hours or less, it is reasonable to assume that it should take only a short amount of time to create a checkpoint. Otherwise, the checkpoint will have little value and it would be more efficient to restart the workunit from the beginning. In addition to this, since the time between checkpoints is on the order of seconds or minutes for the projects about which we were able to get information, the time to create a checkpoint, C, is clearly significantly less than the mean time between failures. Therefore, we use Young's approximation in this work to calculate the optimal checkpoint interval. As values for C, we use 10 seconds and 300 seconds (five minutes) to approximate how long it would take to create a checkpoint. We note that a C of 10 seconds makes the optimal time between checkpoints approximately 26 minutes while a C of 300 makes the optimal time between checkpoints approximately 144 minutes. The time between checkpoints for C of 300 seconds would be appropriate for very long workunits, such as those used by climateprediction.net, while the time between checkpoints for C of 10 seconds would be more appropriate for short workunits used by most volunteer computing projects.

6.3 Simulation Parameter Values

The parameter settings we use for our simulations were:

File Size: 1 KB, 1 MB, 10 MB

Download Speed: 300 kbps, 10 mbps

Delay Bound: 3.5 days, 7 days, 14 days

Checkpoint Cost/Frequency: 10 sec/1574 sec, 300 sec/8620 sec

Task Duration: 2 hours, 4 hours, 8 hours, 16 hours, 24 hours, 48 hours

6.4 Results of Base Parameter Combinations

From the information we gathered on existing volunteer computing projects and the information about checkpointing, we believe that representative parameter settings should be:

File Size: 1 MB

Download Speed: 300 kbps

Delay Bound: 7 days

Checkpoint Cost/Frequency: 10 sec/1574 sec

Task Duration: 4 hours, 24 hours

Those parameter settings are the base values we use for our simulations, although we also did simulations with the other values discussed in Section 6.1. We call the base parameters with 4-hour tasks Baseline1 and the base parameters with 24-hour tasks Baseline2 for ease of referring to them.

We ran simulations using the base parameter settings and all of the task retrieval policies. Table 4 shows how many fewer tasks each policy completed than the number of tasks the *Optimal* policy completed using the base parameter values. The *Download Early* policy yields the best performance of our policies that can be implemented. For 24 hour tasks, the *Download Early* policy provides a significant improvement of 2% more tasks completed over the policies that buffer multiple days of tasks and for 4 hour tasks, the *Download Early* policy provides an improvement of 0.9% more tasks completed over the policies that buffer seven or 14 days of tasks.

Table 4 - Percent Fewer Tasks Completed than Optimal Policy for Base Parameter Settings

Task Completion Time	Download Speed	Buffer None	Download Early	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
4 hours	300 kbps	0.22%	0.03%	0.19%	0.25%	0.46%	0.94%	0.94%
24 hours	300 kbps	0.43%	0.38%	0.86%	1.13%	2.52%	3.33%	3.33%

6.5 *Effects of Varying the Parameters*

We attempted to determine which parameters had a significant impact on the number of tasks the computers completed. To perform this analysis, we used the two base parameter values, Baseline1 and Baseline2 and determined the number of tasks the computers would complete given those parameters. This served as baseline results. Once we had established the baseline results, we varied one parameter at a time and determined the number of tasks the computers would complete using those parameters. We looked to see what effect varying the parameters would have on the number of tasks completed by the 140 computers. We varied the file size from the base value of 1 MB to 1 KB and 10 MB. We varied the download speed from the base value of 300 kbps to 10 mbps. We varied the delay bound from the base value of 7 days to 3.5 days and 14 days and we varied the checkpoint cost and frequency from the base values of 10 seconds and 1574 seconds respectively to 300 seconds and 8620 seconds. A summary of the results of our tests are shown in Table 5, while the full results are show in Table 6 and Table 7.

As one would expect, we found that the task completion time differences produced large differences in the number of tasks completed and have omitted that from the summary table. We found that the impact of decreasing the delay bound is significant for 24-hour tasks, changing the number of tasks completed by up to 9.42 %. However, increasing the delay bound for 24-hour tasks results in a much less significant change in the number of tasks completed (≤ 3.16 %). Adjusting the delay bound has an insignificant impact for 4-hour tasks, resulting in a change of ≤ 1.32 % difference in the number of tasks that are completed. Adjusting the file size and the download speed have a minimal impact on the number of tasks that completed for both 4-hour and 24-hour tasks. However, we speculate that if the download speed were changed to that of a dial-up modem, the effect may be more pronounced. Adjusting the checkpoint cost (and thus frequency) causes significant changes in the number of 4-hour and 24-hour tasks that are completed. In a fixed amount of time, less time is spent creating checkpoints that take 10 seconds to create than is spent creating checkpoints that take 300 seconds to create. In addition to less CPU time being spent creating checkpoints in a fixed amount of time, the progress of the client is being saved more often so less work needs to be repeated in the event of a volunteer computing client being halted. Less CPU time spent creating checkpoints and less work being repeated when clients are restarted after being halted results in significantly more tasks being completed.

Table 5 – Percent Change in Amount of Tasks Completed by Varying Parameters from Baseline Parameters

	Baseline1	Baseline2
File Size Impact	≤ 1.72 %	≤ 0.38 %
Download Speed Impact	≤ 0.18 %	≤ 0.05 %
Delay Bound Impact	≤ 1.32 %	≤ 9.42 %
Checkpoint Cost Impact	≤ 4.66 %	≤ 8.77 %

Table 6 - Effects of Adjusting Parameters for Baseline1 Values (% Change from Baseline1 Values)

	File Size	Delay Bound	Task Completion Time	Download Speed	Checkpoint Cost	Buffer None	Download Early	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
Baseline	1 MB	1 week	4 hours	300 kbps	10 sec							
	1 KB	1 week	4 hours	300 kbps	10 sec	0.18	0.00	0.00	0.00	0.02	0.01	0.01
	10 MB	1 week	4 hours	300 kbps	10 sec	1.72	0.03	0.01	0.08	0.25	0.07	0.07
Baseline	1 MB	1 week	4 hours	300 kbps	10 sec							
	1 MB	1/2 week	4 hours	300 kbps	10 sec	0.30	0.34	0.39	0.59	1.32	0.84	0.84
	1 MB	2 weeks	4 hours	300 kbps	10 sec	0.04	0.04	0.16	0.19	0.37	0.83	0.46
Baseline	1 MB	1 week	4 hours	300 kbps	10 sec							
	1 MB	1 week	2 hours	300 kbps	10 sec	100.44	100.77	101.05	101.01	101.24	101.68	101.68
	1 MB	1 week	8 hours	300 kbps	10 sec	50.36	50.37	50.40	50.43	50.49	50.67	50.67
	1 MB	1 week	16 hours	300 kbps	10 sec	75.57	75.59	75.61	75.67	75.83	75.98	75.98
	1 MB	1 week	24 hours	300 kbps	10 sec	84.00	84.02	84.07	84.10	84.30	84.35	84.35
	1 MB	1 week	48 hours	300 kbps	10 sec	93.44	93.54	97.85	93.75	93.73	93.70	93.70
Baseline	1 MB	1 week	4 hours	300 kbps	10 sec							
	1 MB	1 week	4 hours	10 mbps	10 sec	0.18	0.00	0.00	0.00	0.02	0.01	0.01
Baseline	1 MB	1 week	4 hours	300 kbps	10 sec							
	1 MB	1 week	4 hours	300 kbps	300 sec	4.66	4.66	4.54	4.53	4.52	4.66	4.66

Table 7 - Effects of Adjusting Parameters for Baseline2 Values (% Change from Baseline2 Values)

File Size	Delay Bound	Task Completion Time	Download Speed	Checkpoint Cost	Buffer None	Download Early	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
1 MB	1 week	24 hours	300 kbps	10 sec							
1 KB	1 week	24 hours	300 kbps	10 sec	0.05	0.00	0.00	0.00	0.00	0.00	0.00
10 MB	1 week	24 hours	300 kbps	10 sec	0.38	0.05	0.05	0.05	0.06	0.11	0.11
1 MB	1 week	24 hours	300 kbps	10 sec							
1 MB	1/2 week	24 hours	300 kbps	10 sec	1.67	1.78	9.42	6.57	6.16	5.38	5.38
1 MB	2 weeks	24 hours	300 kbps	10 sec	0.92	0.92	1.08	1.68	3.14	3.16	2.44
1 MB	1 week	24 hours	300 kbps	10 sec							
1 MB	1 week	2 hours	300 kbps	10 sec	1152.53	1156.22	1162.12	1164.51	1181.40	1188.62	1188.62
1 MB	1 week	4 hours	300 kbps	10 sec	524.89	525.69	527.76	529.08	536.76	538.96	538.96
1 MB	1 week	8 hours	300 kbps	10 sec	210.18	210.50	211.36	211.83	215.24	215.21	215.21
1 MB	1 week	16 hours	300 kbps	10 sec	52.64	52.72	53.08	53.07	53.94	53.50	53.50
1 MB	1 week	48 hours	300 kbps	10 sec	59.00	59.56	86.47	60.66	60.10	59.77	59.77
1 MB	1 week	24 hours	300 kbps	10 sec							
1 MB	1 week	24 hours	10 mbps	10 sec	0.05	0.00	0.00	0.00	0.00	0.00	0.00
1 MB	1 week	24 hours	300 kbps	10 sec							
1 MB	1 week	24 hours	300 kbps	300 sec	6.03	6.03	6.66	6.57	7.59	8.77	8.77

Although we have presented the results for the parameters we believe would most representative of those in volunteer computing projects, we believed it was important to test many settings that might be encountered. We present the overall results of those tests here in Table 8. The results show that the *Download Early* policy achieves the best performance of the policies that can be implemented and completes only 0.22% fewer tasks than the optimal policy does. Therefore, we do not believe that attempting to devise adaptive policies to try to close the performance gap could produce a significant performance gain over the *Download Early* policy. We also see that the policies that buffer multiple days of tasks do complete almost a full percent fewer tasks than the *Download Early* policy. This performance difference when multiplied by the millions of computers running BOINC-based volunteer computing projects [122] and the other projects becomes significant.

Table 8 – Fewer Tasks Completed than Optimal Policy (%)

Download Early	Buffer None	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
0.220	0.672	0.897	0.598	1.075	1.316	1.453

We also show what the results would be in the best and worst case scenarios. In the best case scenario, tasks would require small files, Internet connections would be fast, and the delay bound would be high, giving a client more time to complete tasks. In the worst case, the files required would be large and downloaded over a slow connection and the delay bound would be small, forcing tasks to be aborted sooner and leading to more tasks being downloaded. One can make an argument that tasks that take a long time to complete would be the worst, because more will be aborted, leading to fewer completed tasks. One could also argue that tasks that require a short amount of time to complete are worse because while more tasks may be completed, more of them will be downloaded and that will increase the bottleneck effect of the network. Therefore, we present four different sets of results in Table 9 through Table 12 rather than presenting two sets of results. The first set of results, presented in Table 9, assume that in the best case scenario, tasks are long. The second set of results, presented in Table 10, assume that in the best case scenario, tasks are short. The third set of results, presented in Table 11, assume that in the worst case scenario, tasks are long. The fourth set of results, presented in Table 12, assume that in the worst case scenario, tasks are short. The best cases have a file size of 1 KB, a download speed of 10 mbps, a delay bound of 2 weeks, and a checkpoint cost of 10 seconds. The worst cases have a file size of 10 MB, a download speed of 300 kbps, a delay bound of 1/2 week, and a checkpoint cost of 300 seconds. Table 9 shows that the *Download Early* and *Buffer None* policies are the best ones to use in the best case parameter settings with long tasks. For the best case parameters with short tasks, Table 10 shows that the *Buffer None* and *Buffer 1 Task* policies are the best to use. Table 11 shows that the *Download Early* policy is the best one to use with worst case parameters with long tasks. Table 12 shows that the *Buffer None* policy is the best to use with worst case parameters and short tasks. In this case, the *Download Early* policy did worse than all the other policies, likely due to all of the wasted CPU cycles from frequent and lengthy downloads. In all cases, the *Buffer None* policy outperformed the policies that buffered at least 1 day of tasks. We do not expect to see these best and worst case scenarios frequently in real volunteer computing projects, however.

Table 9 – Fewer Tasks Completed than Optimal Policy (%) for Best Case Parameters with Long Tasks

Download Early	Buffer None	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
0.33	0.33	0.44	0.44	0.44	1.44	2.21

Table 10 - Fewer Tasks Completed than Optimal Policy (%) for Best Case Parameters with Short Tasks

Download Early	Buffer None	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
0.03	0.01	0.01	0.03	0.05	0.08	0.24

Table 11 - Fewer Tasks Completed than Optimal Policy (%) for Worst Case Parameters with Long Tasks

Download Early	Buffer None	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
9.97	11.94	70.21	13.78	13.78	13.78	13.78

Table 12 - Fewer Tasks Completed than Optimal Policy (%) for Worst Case Parameters with Short Tasks

Download Early	Buffer None	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
3.82	0.26	0.29	0.60	1.28	1.28	1.28

6.6 Networking Effects on Simulation Results

Although our simulator was carefully designed and constructed, there are two networking values that we did not incorporate. We did not consider packet delay and packet loss in our simulations. Packet delay could affect our results by causing a volunteer computing client to waste some of the available CPU cycles it has while the client is waiting to get a task to perform from the server. Wasting CPU cycles will cause the task to be completed later than it would if those CPU cycles could be used to process the task. In extreme cases, wasted CPU cycles might result in a task not being completed at all or fewer tasks being completed in a fixed amount of time, which would affect our results. In the case where packet delay does not result in wasted CPU cycles, it will not affect the number of tasks completed and thus it will not affect our results. In order to determine the effect that realistic values of packet delay and packet loss would have on our simulation results, we need to be able to calculate the amount of donated CPU time that would be lost to data transfer due to the packet delay and packet loss. The lost time is based on the number occasions that the client must exchange information with the server and the delay and loss rates. In the following sections, we discuss the effects that packet delay and packet loss would likely have on our results.

6.6.1 Data Transfer Information

The instances where packet delay might result in wasted CPU cycles are when the client has no tasks to perform for the project and must get a task. A client may have no tasks after it has completed or aborted the only task it has to perform or as it is starting and has not yet retrieved a task from the server. When a client needs a task, it exchanges messages with the server and downloads a task. In the case where a client completes a task, it must exchange additional messages with the server to return the result of the task. The messages that a client and server will exchange that may cause wasted CPU cycles are shown in Figure 10. We simplify the calculations by observing that all of the communication that takes place can be done in just a small number of packets, except the file download. Each message shown in Figure 10 other than the file download requires only 1 data packet once the client has established a connection to the server. Thus, the packet delay and loss will have a minimal effect on everything other than file downloads and we only consider the effect that packet delay and packet loss have on file downloads when we explain what the effects of packet delay and packet loss have on our results.

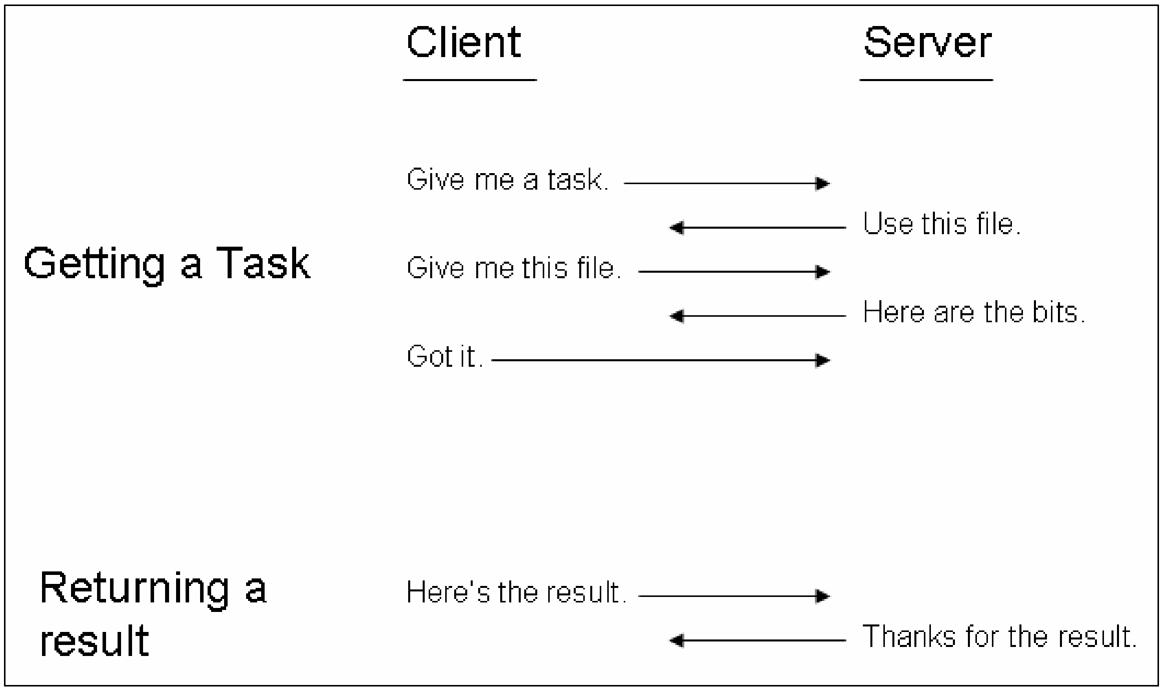


Figure 10 - Messages Exchanged by Volunteer Computing Client and Server for Each Task

6.6.2 Networking Background Information

In order to calculate the effect that packet delay and packet loss will have on our simulation results, we needed to be able to estimate the amount of time that it takes a client to download a file. We use the model from Cardwell, Savage, and Anderson in [123] to determine how long it takes to download a file. This model is unique because unlike the well-known model Padhye, Firoiu, Towsley, and Kurose present in [124] which is only valid for the congestion-avoidance phase of TCP, this model also is valid for the slow-start phase of TCP [123]. Unlike the model from [124] that assumes all data transfer takes place during the congestion-avoidance phase, this model determines the time spent in the slow-start phase of TCP and then uses the model from [124] to determine the time required to transfer the remaining portion of the file that was not transferred during the slow-start phase [123]. The parameters in the model from [123] are:

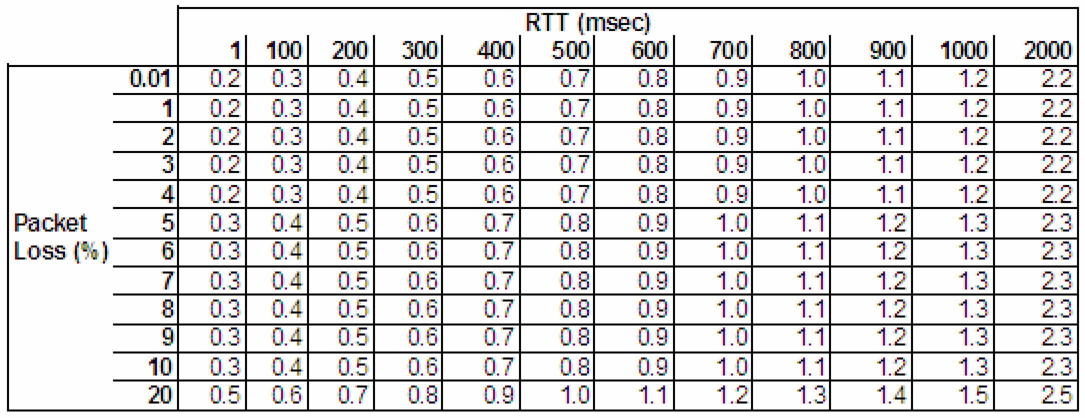
- W_{max} – The maximum size of the congestion window.
- b – The number of packets acknowledged with a single ACK.
- T_0 – The amount of time the sender waits for an ACK. If the sender does not receive an ACK by this time, it retransmits the packet.
- $E[T_{delack}]$ – The amount of time before the delayed ACK timer goes off.
- RTT – The average round trip time of packets.
- p – The probability that a packet is lost.

In our calculations, we use W_{max} = 12 segments for the maximum congestion window size because Windows XP uses this as its default congestion window size [125]. We assume that b equals 2, mimicking the delayed ACK method used by many TCP implementations, as suggested by [124]. We use T_0 = 1 second. We use 200 msec for $E[T_{delack}]$, as Windows XP uses this value [125]. RTT and p are the parameters we test. We examine the time to download a file for values of RTT = 1, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, and 2000 msec, p = .0001, .01, .02, .03, .04, .05, .06, .07, .08, .09, .10, and .20, and file sizes of 1 KB, 1 MB, and 10 MB.

6.6.3 Results

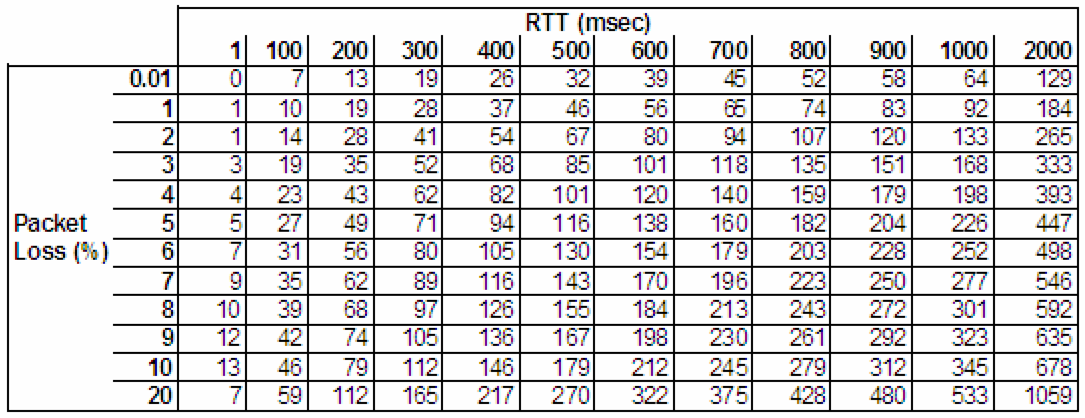
We found that for each file size, the effect of increasing the round trip time (RTT) linearly while holding the packet loss rate constant increased the amount of time it took to download the file in an approximately linear manner. Holding the RTT constant while increasing the packet loss rate linearly results in the amount of time it takes to download a file increasing in a slightly less linear manner, with a bigger increase in download time coming in the first few increases in packet loss. The results of our calculations about the effects of RTT and packet loss are shown in Table 13, Table 14, and Table 15.

Table 13 - Time (seconds) to Download a 1 KB File



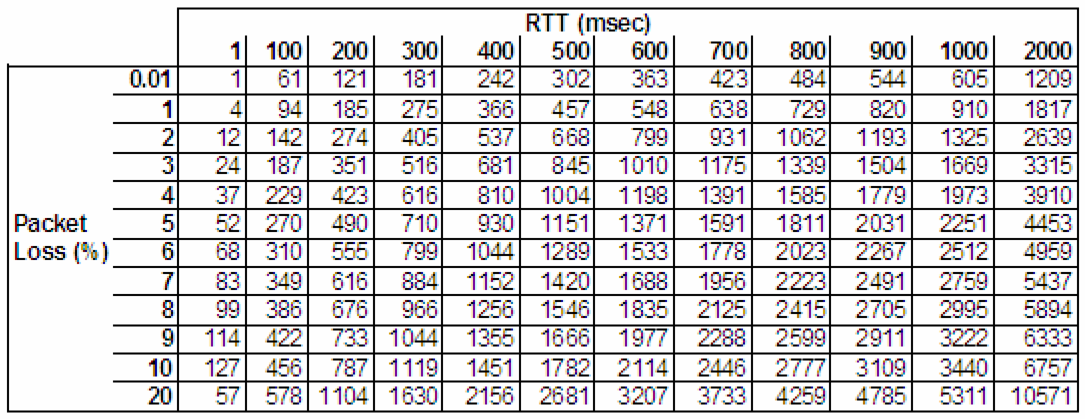
		RTT (msec)											
		1	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**	**2000**
	0.01	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	2.2
	1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	2.2
	2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	2.2
	3	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	2.2
	4	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	2.2
Packet	**5**	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	2.3
Loss (%)	**6**	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	2.3
	7	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	2.3
	8	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	2.3
	9	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	2.3
	10	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	2.3
	20	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	2.5

Table 14 - Time (seconds) to Download a 1 MB File



		RTT (msec)											
		1	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**	**2000**
	0.01	0	7	13	19	26	32	39	45	52	58	64	129
	1	1	10	19	28	37	46	56	65	74	83	92	184
	2	1	14	28	41	54	67	80	94	107	120	133	265
	3	3	19	35	52	68	85	101	118	135	151	168	333
	4	4	23	43	62	82	101	120	140	159	179	198	393
Packet	**5**	5	27	49	71	94	116	138	160	182	204	226	447
Loss (%)	**6**	7	31	56	80	105	130	154	179	203	228	252	498
	7	9	35	62	89	116	143	170	196	223	250	277	546
	8	10	39	68	97	126	155	184	213	243	272	301	592
	9	12	42	74	105	136	167	198	230	261	292	323	635
	10	13	46	79	112	146	179	212	245	279	312	345	678
	20	7	59	112	165	217	270	322	375	428	480	533	1059

Table 15 - Time (seconds) to Download a 10 MB File



		RTT (msec)											
		1	100	200	300	400	500	600	700	800	900	1000	2000
Packet Loss (%)	0.01	1	61	121	181	242	302	363	423	484	544	605	1209
	1	4	94	185	275	366	457	548	638	729	820	910	1817
	2	12	142	274	405	537	668	799	931	1062	1193	1325	2639
	3	24	187	351	516	681	845	1010	1175	1339	1504	1669	3315
	4	37	229	423	616	810	1004	1198	1391	1585	1779	1973	3910
	5	52	270	490	710	930	1151	1371	1591	1811	2031	2251	4453
	6	68	310	555	799	1044	1289	1533	1778	2023	2267	2512	4959
	7	83	349	616	884	1152	1420	1688	1956	2223	2491	2759	5437
	8	99	386	676	966	1256	1546	1835	2125	2415	2705	2995	5894
	9	114	422	733	1044	1355	1666	1977	2288	2599	2911	3222	6333
	10	127	456	787	1119	1451	1782	2114	2446	2777	3109	3440	6757
	20	57	578	1104	1630	2156	2681	3207	3733	4259	4785	5311	10571

We point out that the only policies we simulated that are affected by the increase in packet delay and packet loss are the policies that do not buffer tasks. This is because in steady state, the policies that do buffer tasks do the downloading of tasks while they still have tasks to work on. Thus, only the *Buffer None* policy and potentially the *Download Early* policy would be affected by an increase in packet delay and packet loss. We look at the effects of packet delay and packet loss on the *Buffer None* policy first and then we examine the effects on the *Download Early* policy. We highlight the effects that packet loss and packet delay would have on our simulation results for 24-hour tasks and acknowledge that the effects would be increased or decreased for tasks that were longer or shorter. We also focus on 10% packet loss and 1 second RTT, which we expect to be a worst case scenario, and allow the reader to perform the simple calculations to understand how less severe network conditions would impact our results.

The *Buffer None* policy wastes CPU time whenever it downloads a task. The most CPU time that will be wasted is equal to (the number of tasks downloaded * the time to download a task). For a computer that is always available to work on volunteer computing projects, at most 28 24-hour tasks would be downloaded in a 28 day period.

For 1 KB files and packet losses of 20% and a 2 second RTT, the wasted time would be 28 tasks * 2.5 seconds = 70 seconds which will have almost no effect at all on the number of tasks a volunteer computing client will complete. A packet loss of 10% and a 1 second RTT would result in 28 * 1.3 = 36.4 wasted seconds. Therefore, the effect of packet loss and packet delay is negligible on the *Buffer None* policy with 1 KB files. With 1 MB files, a 1 second RTT and 10% packet loss would result in 28 * 354 = 9660 wasted seconds or approximately 2.68 wasted hours of donated CPU time in the worst case scenario. This is approximately equivalent to 0.4% of the donated CPU time. 10 MB files with a 10% packet loss and 1 second RTT would result in 28 * 3440 = 96320 seconds of wasted CPU time in the worst case. This is equivalent to almost 27 hours of wasted CPU time, which is about 4% of the donated CPU time that would be wasted.

Because the Download Early policy we simulated begins downloading the next task when it is 95% complete with the task it is working on, it will have a minimum of 5% of the time required to complete a task to download the next task while not affecting the number of tasks that a client completes. For 24-hour tasks, 5% is 1.2 hours (4320 seconds). We point out that for 1 KB, 1 MB, and 10 MB files, the increased packet delay and packet loss never would cause the client running the *Download Early* policy to be downloading a file while it has no task to complete if the tasks are 24-hours. In fact, for 1 KB and 1 MB files, even 20% packet loss and 2 second RTT would not cause a client running the *Download Early* policy to waste any CPU time.

Based on 10% packet loss and 1 second RTT, our simulation results for the *Download Early* policy for 24-hour tasks would not be affected and the results for the *Buffer None* policy would not be affected significantly for 1 KB or 1 MB files. The effects on the *Buffer None* policy are a bit more severe for 10 MB files. Therefore, we are able to continue our recommendation of using the *Download Early* policy for long tasks.

6.6.4 Other Thoughts

We acknowledge that our results may not hold for parameter settings beyond the ones we have tested. A personal computer at a person's house might experience higher packet delays due to several factors. A person's ISP may not have as fast a connection as the computers involved in our measurements. A person may also be connected to their ISP through a wireless network, which can cause higher packet delays based on a variety of conditions, including proximity to the access point and the amount of other wireless traffic in close proximity. Finally, we have assumed that people have high speed Internet connections such as DSL or cable modems, rather than dial-up 56 kbps modems or other Internet connections that run at slow speeds such as a cellular network. We acknowledge using slower connections like these may change the results of our simulations, but we do not believe that cellular networks and 56 kbps modems will be used for volunteer computing. In Section 3.2.1, we explained that people are unlikely to run down the battery of wireless devices such as cell phones and PDAs, and thus we do not expect people to be using cellular networks for volunteer computing. We also suggest that at this point in time, the bulk of the people willing to participate in volunteer computing are those who are very technology savvy and thus have high speed Internet connections. However, very slow connections and very high packet delays will make the *Buffer None* task retrieval policy less effective in comparison to the other policies. This effect will occur because the *Buffer None* policy will have wasted CPU cycles for every task that is downloaded, the other policies will be able to avoid wasting most of those CPU cycles because only the first task that is downloaded by the other policies should incur the wasted CPU cycles. The fact that the other policies download tasks before the current task is completed will prevent the wasting of most of the CPU cycles that the *Buffer None* policy will waste.

6.7 Summary

We have shown the results of our simulations using different task retrieval policies. For the parameters we expect will be most common for volunteer computing projects, we found that our *Download Early* policy provides the best results of the policies that do not require knowledge of the computer's availability beforehand. For 24-hour tasks, the *Download Early* policy provides a 2% improvement over the policies that buffer multiple days of tasks and smaller improvements over the other task retrieval policies. For 4-hour tasks, the *Download Early* policy provides a 0.9% improvement over the policies that buffer at least seven days of tasks and smaller improvements over the other task retrieval policies. We found that the task completion time, delay bound, and checkpoint cost can have a large impact on the number of tasks a computer can complete using the various policies. We found that the file size and download speed did not significantly affect the number of tasks completed.

For the entire set of parameter combinations, the Download Early policy outperformed the other policies that did not require a priori knowledge of a computer's availability. The performance gap is large enough so that millions of computers using it instead of the other policies could lead to large increases of the number of tasks volunteer computing projects can complete. However, the gap between the *Download Early* policy and the optimal policy, which requires a priori knowledge of a computer's availability, is not significant enough to warrant attempting to create complicated policies to attempt to close the gap further. We have explained the effects that introducing packet delay and packet loss have on our results, negatively affecting all of the policies the same way, except for the *Buffer None* policy which suffers more ill effects and reduces its usefulness. Therefore, we recommend that volunteer computing projects use the *Download Early* task retrieval policy.

7 Validation of Simulation Results

In order to validate that the results of our simulations would hold in the real world, we developed a volunteer computing server program and an emulated volunteer computing client to test the task retrieval policies under more realistic conditions. We ran the server program and the emulated volunteer computing client with different policies using a subset of the traces we collected.

7.1 Experimental Setup

We used a private network of 52 computers to run the validation. One computer ran nothing but the volunteer computing server program and a web server from which the clients downloaded data files. A second computer functioned as a router. The other 50 computers ran the emulated volunteer computing clients. We used three types of computers to run the emulated clients: 10 computers with Celeron 333 MHz CPUs, 20 computers with Pentium II 350 MHz CPUs, and 20 computers with Celeron 400 MHz CPUs. All the computers had 128 MB of RAM and ran Windows XP and the Java Runtime Environment 1.5.0_09 from Sun Microsystems. Figure 11 illustrates our setup.

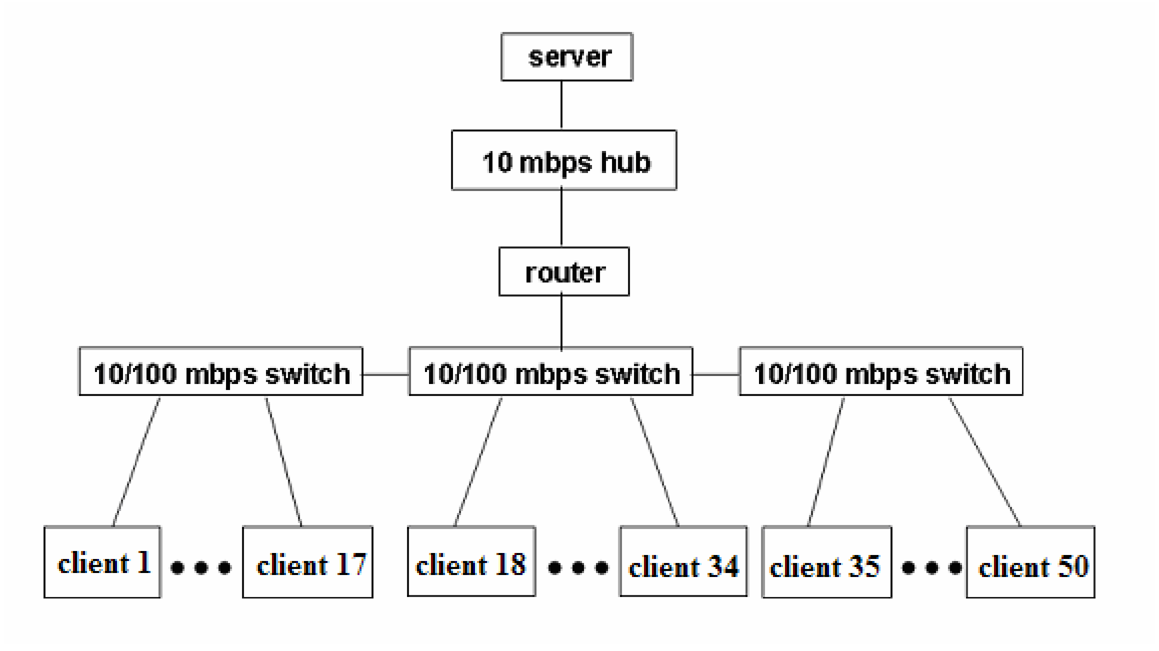


Figure 11 - Validation Experimental Setup

A variety of parameters were involved in the validation including which parameter settings from the simulations to test, which task retrieval policies to test, which traces to use, and the network conditions. Due to our limited equipment, we knew we would only be able to test some subset of the task retrieval policies, parameter settings, and traces. Our volunteer computing server was extremely lightly loaded. We monitored the CPU utilization at various random points and it was always 0%. We also point out that as discussed in Section 3.2.2, Anderson et al. showed that one computer that cost about \$4000 in 2005 could support clients retrieving 8.8 million tasks per day, so our server which would hand out fewer than 1000 tasks per day in the worst case was clearly underutilized [76]. Thus, the 10 mbps network connection between the clients and server would be the bottleneck of our experiment, allowing us to run different policies in the validation at the same time without affecting the trials.

7.1.1 Parameter Settings

For the validation, we elected to use the parameter settings that we previously determined were the ones we expected to see most commonly in real volunteer computing projects, as discussed in Section 6.4. This gave us a 1 MB file size, a delay bound of seven days, a checkpoint cost of 10 seconds and a checkpoint frequency of 1574 seconds. We needed to select a download speed from 300 kbps and 10 mbps and a task duration from four hours and 24 hours. We selected the download speed of 10 mbps because this was easier to implement in our experiment. We selected the task duration of 24 hours because we believe our experience with volunteer computing projects is that tasks take an amount of CPU time much closer to 24 hours than to four hours.

7.1.2 Task Retrieval Policies

We discussed four task retrieval policies that can be implemented (*Buffer None*, *Download Early*, *Buffer One Task*, and *Buffer N Days*) and the task retrieval policy that is theoretical (*Optimal*) in Section 5.1.3. We tested all four of the policies that can be implemented. Due to our limited hardware, for the *Buffer N Days* policy, we were forced to select only two of the four values of *N* that we had simulated. We selected one day and seven days for the *Buffer N Days* policy, electing not to test one day and 14 days. With the parameter settings we elected to use, the numbers of tasks completed are identical when seven days of tasks and 14 days of tasks are buffered, but the number of completed tasks differs from the results of buffering one day and 3.5 days of tasks. We chose to test buffering one day of tasks instead of 3.5 days of tasks because we felt that users were more likely to buffer one day of tasks instead of 3.5 days of tasks if they were participating in a volunteer computing project created with the BOINC framework.

7.1.3 Trace Selection

Due to the limited hardware we had at our disposal, we were only able to run the validation for a small number of traces. In order to determine which traces to use for our validation, we examined the number of tasks completed for each task retrieval policy with every trace, given the parameter settings we chose to test. We identified all the traces where the task retrieval policies we elected to test did not produce the same number of completed tasks because our simulations highlighted differences between policies in those cases and we wanted to test if that would hold true under the more realistic conditions of the validation. This narrowed the list of traces that we had to choose from down to 41. From the 41 remaining traces, we removed the traces from the computers that had very little availability (< 15%), because these traces would show the least interesting information, leaving 35 traces. Of the 35 traces, we selected the traces that resulted in the greatest variability in the number of tasks completed for the simulations, leaving 12 traces. Of the remaining 12 traces, we randomly selected the 10 traces we would test:

1. business_trace_9.trc
2. business_trace_26.trc
3. ccc_trace_7.trc
4. ccc_trace_10.trc
5. ccc_trace_42.trc
6. ccc_trace_55.trc
7. ccc_trace_64.trc
8. home_trace_1.trc
9. home_trace_24.trc
10. student_trace_4.trc

7.1.4 Task Creation

A key part of volunteer computing is the tasks that the clients perform. We devised a number crunching task that we believe is similar in nature to tasks other volunteer computing clients would do. Our task was performing Fast Fourier Transforms (FFTs) using input values from the file the client downloaded. We needed to devise a 24-hour task, so we ran our validation client for one hour, having it do as many FFTs as it could and reporting that value. For each of the three hardware configurations that we would use in the validation (Dell Optiplex G1s with Celeron-400 MHz CPUs, Celeron-333 MHz CPUs, and Pentium II-350 MHz CPUs), we ran 50 trials per computer for as many computers as we were able to at the time (24 Celeron-400 MHz systems, 10 Pentium II-350 MHz systems, and 9 Celeron-333 MHz systems). We took the average of the results for each class of CPU and scaled it up to the number of FFTs that a computer should be able to complete in 24 hours to get our task for each of the three hardware configurations. Thus, a 24-hour task on a computer with a Celeron-400 MHz CPU consisted of more FFTs than a 24-hour task for a computer with a Celeron-333 MHz CPU.

7.1.5 Volunteer Computing Server Program

The volunteer computing server program is a multithreaded Java program (the source code is in the validator.zip file that is available where you downloaded this dissertation). The server program spawns a new thread to handle each incoming connection request. The incoming connections ask for a task, acknowledge the successful download of a file, or return the result of a task. When a request comes in for a task, the server assigns the next task in its pool of tasks and responds with a file name. The client downloads the specified file from the web server and sends an acknowledgement to the server. The abort time for the task is based on the time of the acknowledgement.

7.1.6 Emulated Volunteer Computing Client Program

The volunteer computing client is also a multithreaded Java application (source code in Appendix D). When the client starts, a single thread (the *control thread*) starts. The *control thread* sleeps for some number of minutes equal to 5 * the client's unique identification number, which is a command line argument. This initial sleeping period staggers the start times of the clients so they are not all downloading the initial files at exactly the same time, making the situation more realistic. Once the client has finished its initial sleep, it wakes up and reads a trace file specified as a command line argument. The trace file is one of the 140 traces we collected and discussed in Chapter 4. The data in the trace file is used by the *control thread* to instruct other threads to start, pause, resume, or stop, based on the trace at the appropriate times to mimic the status of the computer from which the trace was collected. The other threads are a *science application thread*, a checkpoint thread, and one or more *download threads*. Those three threads form the actual volunteer computing client, while the *control thread* simulates the computer and screensaver status. The *control thread* runs for the entire duration of the trace, starting, stopping, pausing, and resuming the *science application thread* and *download threads* at the appropriate times. The *science application thread* controls the *checkpoint thread*. Once the end of the trace has been reached, the client outputs the number of tasks it has completed and terminates.

7.2 *Results of Validation*

A comparison of the number of tasks that the clients completed in the simulation and validation are shown in Table 16.

Table 16 - Comparison of Number of Tasks Completed by Validation and Simulation

		Task Retrieval Policy				
Trace	**Method**	**Download Early**	**Buffer None**	**Buffer 1 Task**	**Buffer 1 Day**	**Buffer 7 Days**
Business 9	Simulation	12	12	11	11	11
	Validation	11	12	10	9	10
Business 26	Simulation	16	16	15	15	15
	Validation	16	16	15	15	12
CCC 7	Simulation	16	16	16	16	13
	Validation	16	16	15	15	12
CCC 10	Simulation	16	16	16	16	14
	Validation	16	16	15	15	12
CCC 42	Simulation	17	17	17	17	15
	Validation	16	16	15	16	16
CCC 55	Simulation	20	20	20	20	18
	Validation	20	20	19	20	17
CCC 64	Simulation	22	22	22	22	20
	Validation	22	22	21	21	19
Home 1	Simulation	9	9	8	8	8
	Validation	10	10	8	8	7
Home 24	Simulation	10	10	8	8	8
	Validation	10	10	9	8	8
Student 4	Simulation	11	11	11	11	9
	Validation	11	11	11	11	8

We observed that the number of tasks completed by a client during validation is fewer than the number of tasks completed by a simulated client almost half of the time. We believe that this may be due to several factors. One reason that may account for the difference between the number of tasks completed by the emulated clients and the simulations is the difference between the tasks that the simulations and emulated clients in the validation used. The simulations assumed that a task required some number of seconds to be completed. Every time the clock in the simulation advanced one second and the client was available for volunteer computing, the number of seconds left to complete the task was decremented by one. However, the emulated clients were given realistic tasks consisting of a fixed number of Fast Fourier Transforms (FFTs). The number was derived from observing how many FFTs the computers could do in a one hour (we used the average from hundreds of tests for each CPU speed used in the validation) and scaled up to the number the computer should be able to do in 24 hours. Because the computers did not perform the same number of FFTs each time in an hour, completing a task would not take exactly the same amount of time (exactly 24 hours) in the validation the way it did in the simulation. The variation in the exact amount of time a task took to be completed coupled with the imprecision of timers in Java [126] could have led to checkpoints being created later in the validation than in the simulation. If the creation of a checkpoint almost failed in the simulation and was started later in the validation, then the creation of that checkpoint may have failed in the validation even though it succeeded in the simulation. In this case, approximately 1.82% of the CPU time required to complete a task would have been wasted. 54 checkpoints are created during the processing of one task. It is possible that the creation of one or more of these checkpoints could have failed during the processing of a task in the validation but succeeded in the simulation, due to the skew caused by the timers and because it does not take exactly the same amount of time to complete every task. In the case that one checkpoint per task processed by the emulated client failed because of the skew, that could result in an emulated client completing 18% to 36% less of one task than the simulated client completed. In some cases, the failure to create a checkpoint could result in a task being aborted by the emulated volunteer computing client because the task was not completed on time, which could result in the number of tasks completed by the emulated client and simulated client differing by an entire task or more.

7.3 Summary

We have discussed our methodology for validating that our simulations are accurate. We ran a volunteer computing server on one server and emulated volunteer computing clients on 50 other computers. We tested five task retrieval policies with 10 computer usage traces. The number of tasks completed by each emulated volunteer computing client are very similar to the number of tasks that our simulator predicted the clients would complete for the given parameter settings. Therefore, we are confident that our simulation results are accurate.

8 Effects of Multiple Types of Clients

8.1 Discussion

In order to increase the productivity of volunteer computing projects, it is important to write the volunteer computing clients in the manner that allows the most work to be completed by the volunteered computers. This idea is composed of two pieces: the language in which volunteer computing clients are implemented and the types of programs that volunteer computing clients are produced as. Because new programming languages are developed, the speed of languages changes relative to each other, and the popularity of programming languages changes, we do not spend much time discussing the choice of a specific language used to implement volunteer computing clients. However, we do point out that the language that enables the most work to be completed may not necessarily be the language that executes the same set of operations the fastest. Clients written in some languages may provide advantages over clients written in other languages, such as convenience and security. Thus, people may be more likely to participate in volunteer computing projects based on the programming language in which the client is written. In this case, a slower client may accomplish more work than a faster client if the slower client entices more people to participate.

Currently, volunteer computing clients are written in two forms: screensavers and service (Windows)/daemon (Linux & Unix) processes. We point out that it is also possible to create web-based clients, such as Java applets. The different types of clients offer different feature sets, making some types of clients more attractive to certain users than others. For example, screensaver clients do not affect computer performance when a person is actively using a computer, except during long non-interactive jobs such as a virus scan. Clients that run as a service/daemon process are able to complete more tasks than clients that run as a screensaver. A web-based client like a Java applet also can run like a background process, but can provide more security than other types of clients. Because clients that run constantly in the background can likely complete more tasks than clients that run as a screensaver, assuming both clients are written in the same language, we believe it is important for us to give some estimate of how much more work can be accomplished by clients that run constantly.

8.2 Methodology

We modified our simulator to allow us to observe how many more tasks clients that run as services would complete than clients that run as screensavers. It is not possible to determine what percentage of CPU time a service/daemon process will receive at any given time and what the effect of operating system features such as caching (and thus cache misses that would occur) and page faults would be. Therefore, our modified simulations assume that the client receives 100% of the CPU cycles whenever the computer is powered on, even when the screensaver is not running. We acknowledge that this provides an overestimate of the number of tasks that a service/daemon client will complete. We leave it to the reader to extrapolate how many tasks the service/daemon client would complete and point out that it will be more than the number of tasks completed by the screensaver, but less than the number of tasks that our simulations of a service/daemon clients completed.

8.3 Simulation Results

We show the results of the simulations of a service/daemon volunteer computing client for the two sets of base parameter values we believe will be the most common in volunteer computing projects, as discussed in Section 6.4. We show the number of times more tasks completed by each policy by the service/daemon than by the screensaver client in Table 17. For all of the task retrieval policies, the service/daemon clients completed approximately 1.5 as many tasks as the screensaver clients completed. This improvement of 50% is significant enough that this method should be explored further.

Table 17 - Number of Times More Tasks Completed by the Service/Daemon than by the Screensaver

Task Duration	Download Speed	Optimal	Buffer None	Download Early	Buffer 1 Task	Buffer 1 Day	Buffer 3.5 Days	Buffer 7 Days	Buffer 14 Days
4 hours	300 kbps	1.52	1.52	1.52	1.52	1.52	1.52	1.53	1.53
24 hours	300 kbps	1.56	1.54	1.54	1.53	1.55	1.57	1.57	1.57

8.4 Summary

Our simulations showed a significant increase of approximately 50% in the productivity of volunteer computing clients (approximately a 50% increase in the number of tasks that were completed) when they were run as service/daemon processes instead of as screensavers. Given that often only a fraction of the CPU is used when a personal computer's screensaver is not running, we expect that this is a reasonably accurate estimate of the performance improvement that can be gained by running clients as service/daemon processes instead of as screensavers. Therefore, we encourage the creators of volunteer computing projects to find ways to encourage volunteers to run service/daemon clients instead of screensaver clients if possible. We also point out that with multi-core CPUs becoming standard, service/daemon clients will be even more productive.

9 Conclusions

We have made a number of contributions to the field of volunteer computing in this dissertation. Using simulations, we have evaluated the two task retrieval methods (*Buffer None* and *Buffer N Days*) that are used in different volunteer computing projects to determine which method results in higher productivity of volunteer computing projects. In our simulations, *Buffer None*, which is the task retrieval method that is not often used resulted in more tasks being completed than *Buffer N Days*, the predominantly used task retrieval method. We have examined the way task retrieval methods work in volunteer computing projects and proposed two additional task retrieval methods (*Download Early* and *Buffer 1 Task*) in an attempt to create methods that result in even higher productivity for volunteer computing. For our two baseline parameters, our simulations showed that the *Download Early* method resulted in 0.91% and 2.95% more tasks being completed than *Buffer N Days* for N=7 and N=14. The Download Early method resulted in more tasks being completed than the *Buffer None* method, too. We have developed an optimal policy that established an upper bound on the number of tasks that can be completed and we showed that the difference in the number of tasks completed by clients using the Download Early task retrieval policy and the Optimal policy is so small that it does not make sense to try to develop an adaptive policy to increase the productivity of volunteer computing. We recommend that people who implement volunteer computing projects set their clients to use our *Download Early* task retrieval policy, rather than the current standard *Buffer N Days* policy. During our evaluation of the different policies, we observed that varying our simulation parameters had little effect on the number of tasks that were completed, with the exception of the delay bound. For 24-hour tasks, increasing or decreasing the delay bound results in a significant increase or decrease in the number of tasks that are completed.

We have explored the increase in productivity of volunteer computing projects if volunteer computing clients ran constantly in the background as a service/daemon process instead of running as a screensaver. We found that the number of completed tasks increased by approximately 50% if the client was run as a service/daemon process rather than as a screensaver.

We have explored the potential benefit of using video game consoles to run volunteer computing clients. Although our tests with the Sony PlayStation2 showed that the PlayStation2 was far from an ideal platform for volunteer computing, Sony and Folding@home have explored using the PlayStation3 for volunteer computing and found that it is an excellent platform.

We have collected 140 traces of computer usage activity for 28 days from home, business, public access, and undergraduate student computers. We have made these traces publicly available for others to use. We have also compared the distribution of computer availability of Derrick Kondo's traces from the San Diego Supercomputer Center to our most closely related traces. We found that the distributions of availability between our data set and his data were not alike.

We have collected publicly available information about real values used for volunteer computing projects and combined it into the first summary of volunteer computer project values of which we are aware.

10 Future Work

There are several ways to extend the work in this dissertation.

1. We believe that the most effective way to improve the productivity of volunteer computing is to significantly increase the number of participants. In order to increase participation, it is important to understand why such a small percentage of Internet-connected computers are used for volunteer computing [37]. We believe that a comprehensive study to determine why so many people do not participate in volunteer computing should be done. The results of the study may then be used to influence implementation of volunteer computing clients or spur future research required to make people more willing to participate.

2. Another direction for future work is to determine the ramifications of implementing volunteer computing clients as unsigned Java applets. Because unsigned applets could provide a level of security that can't be provided by current clients that run as a screensaver or as a service/daemon process, clients implemented as applets might motivate more people to participate in volunteer computing. However, the potential participation increase needs to be weighed against the potential costs. One concern about applets is that unlike a screensaver or a service/daemon process that starts automatically, running a volunteer computing client that is implemented as a Java applet requires the volunteer to open a web browser and navigate to a web page to start the client. This inconvenience may result in fewer donated CPU cycles, as people may forget to activate the client every time they start their computer. One more drawback to implementing

volunteer computing clients as unsigned Java applets is that the ability of the clients to checkpoint is severely restricted. The only way for the clients to checkpoint would be to save their checkpoint files on the server from which the applet was downloaded. If applets are to be a potential implementation, future work needs to determine how many clients could be supported by a single server so that the costs are better understood.

3. The work in this dissertation has focused on volunteer computing in the context of clients running on single CPU computers. With the trend of putting multiple cores on a die so that parallel processing can be done in commodity computers, it is important to understand how to take advantage of the extra processing power. We suggest that it would make sense to determine whether it is better to process multiple workunits at one time on a multi-core computer or whether running a single workunit that is multithreaded will increase the productivity of volunteer computing more.

4. Another direction for future work is to understand what role, if any, processor affinity should play in volunteer computing clients. The importance of processor affinity may be directed affected by whether volunteer computing clients continue to be implemented to process single work units at a time or multiple workunits at once.

11 References

1 D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. "SETI@home: An Experiment in Public-Resource Computing", *Communications of the ACM*, vol. 45, pp. 56-61, Nov. 2002, [Online] Available: http://setiathome.ssl.berkeley.edu/cacm/cacm.html.

2 BOINC staff, "Choosing BOINC Projects", http://boinc.berkeley.edu/projects.php, Modified 12/10/07, Accessed 2/9/08.

3 D. Anderson, "A System for Public-Resource Computing and Storage", *Proc. 5th IEEE/ACM International Workshop on Grid Computing*, 2004, [Online] Available: http://boinc.berkeley.edu/grid_paper_04.pdf.

4 GIMPS staff. "GIMPS Home Page", http://www.mersenne.org/prime.htm, Modified 9/11/06, Accessed 7/11/07.

5 A. Patrizio, "Codebusters Crack Encryption Key", http://www.wired.com/science/discoveries/news/2002/10/55584, Modified 10/7/02, Accessed 2/12/08.

6 A. Patrizio, "56 a Bit Short of Secure", http://www.wired.com/science/discoveries/news/2000/01/33695, Modified 1/17/00, Accessed 2/12/08.

7 distributed.net, "stats.distributed.net – Statistics Server", http://stats.distributed.net/, Modified 5/13/04, Accessed 2/12/08.

8 M. Karaul, "Metacomputing and Resource Allocation of the World Wide Web", Ph.D. Thesis, New York University, 1998, [Online] Available http://cs.nyu.edu/milan/publications/karaul-thesis.pdf.

9 T. Brecht, H. Sandhu, M. Shan, and J. Talbot, "ParaWeb: Towards Word-Wide Supercomputing", *Proc. Seventh ACM SIGOPS European Workshop*, pp. 181-188, 1996, [Online] Available: http://bcr2.uwaterloo.ca/~brecht/papers/getpaper.php?file=paraweb.pdf.

10 M. Neary, B. Christiansen, P. Cappello, and K. Schauser, "Javelin: Parallel Computing on the Internet", *Future Generation Computer Systems*, vol. 15, pp. 659-674, 1999 [Online] Available: www.cs.ucsb.edu/~neary/WORK/fgcs.ps

11 POPCORN staff, "POPCORN Documentation", http://www.cs.huji.ac.il/labs/popcorn/documentation/ Accessed 2/22/05.

12 L. Sarmenta. “Volunteer Computing”, Ph.D. thesis, MIT, 2001, [Online] Available: http://www.cag.lcs.mit.edu/bayanihan/papers/phd/sarmenta-phd-mit2001.pdf.

13 K. Amorin, A. Covati, D. Finkel, M. Lee, and C. Wills, "An Applet-Based Approach to Large-Scale Distributed Computing", *Proc. International Network Conference*, pp. 175–182, 2000, [Online] Available: http://web.cs.wpi.edu/~cew/papers/inc00-applet.doc.

14 W. Wong, “Publications of Weng-Fai Wong”, http://www.comp.nus.edu.sg/~wongwf/publications.html, Modified 1/29/05, Accessed 2/22/05.

15 BOINC staff, “Berkeley Open Infrastructure For Network Computing,” http://boinc.berkeley.edu/, Modified 2/1/05, Accessed 2/9/05.

16 G. Fedak, C. Germain, V. Neri and F. Cappello, “XtremWeb: A Generic Global Computing System”, *Proc. 1st IEEE International Symposium on Cluster Computing and the Grid*, pp. 582-587, 2001, [Online] Available: www.lri.fr/~fedak/XtremWeb/downloads/Gcpd.zip.

17 C. Germain, V. Neri, G. Fedak, and F. Cappello, “XtremWeb: Building an Experimental Platform for Global Computing”, Presented at GRID 2000, 12/00, [Online] Available: www.lri.fr/~cecile/publis/grid2000.ps.gz.

18 XtremWeb staff, “XtremWeb,” http://www.lri.fr/~fedak/XtremWeb/introduction.php3 Modified 2/8/05, Accessed 2/19/05.

19 G. Woltman, “The Mersenne Newsletter”, http://www.mersenne.org/news1.txt, Modified 2/24/96. Accessed 7/13/05.

20 Distributed.net staff, “distributed.net: Projects”, http://www.distributed.net/projects.php, Modified 11/4/04, Accessed 7/13/05.

21 Distributed.net staff, “distributed.net: Project OGR”, http://www.distributed.net/ogr/ Modified 5/31/05, Accessed 7/13/05.

22 Seti@home staff, “Seti@home”, http://setiathome.berkeley.edu/, Modified 7/12/05, Accessed 7/13/05.

23 Seti@home staff, “About SETI@home”, http://setiathome.berkeley.edu/sah_about.php, Accessed 7/13/05.

24 S. Pfalz, “Seti@Home P-Stats V0.34”, http://www.saschapfalz.de/pstats.php, Modified 4/26/05, Accessed 4/27/05.

25 Grid.org staff, “GRID.ORG – About Us:Overview”, http://www.grid.org/about/, Accessed 7/14/05.

26 Grid.org staff, “GRID.ORG – Project Overview”, http://www.grid.org/projects/, Accessed 4/1/05.

27 BOINC Statistics staff, “BOINC Statistics”, http://www.boincstats.com, Modified 4/27/05, Accessed 4/27/05.

28 Distributed.net staff, “OGR-24 / Overall Project Stats”, http://stats.distributed.net/projects.php?project_id=24, Modified 10/31/04, Accessed 4/27/05.

29 Distributed.net staff, “RC5-72 / Overall Project Stats”, http://stats.distributed.net/projects.php?project_id=8, Modified 4/26/05, Accessed 4/27/05.

30 Distributed.net staff, “OGR-25 / Overall Project Stats”, http://stats.distributed.net/projects.php?project_id=25, Modified 4/26/05, Accessed 4/27/05.

31 Distributed.net staff, “RC5-56 / Overall Project Stats”, http://stats.distributed.net/projects.php?project_id=3, Modified 10/2/97, Accessed 4/27/05.

32 Distributed.net staff, “RC5-64 / Overall Project Stats”, http://stats.distributed.net/projects.php?project_id=5, Modified 7/14/02, Accessed 4/27/05.

33 Folding@home staff, “daily_user_summary”, http://vspx27.stanford.edu/daily_user_summary.txt, Modified 4/27/05, Accessed 4/27/05.

34 GIMPS staff, “Remaining Top Producers”, http://www.mersenne.org/top5.htm, Modified 4/26/05, Accessed 4/27/05.

35 Grid.org staff, “GRID.ORG – Grid MP Global Service Statistics”, http://www.grid.org/stats/, Modified 4/27/05, Accessed 4/27/05.

36 Internet Systems Consortium, Inc. “ISC Internet Domain Survey”, http://www.isc.org/index.pl?/ops/ds/, Accessed 3/2/05.

37 J. Bohannon, “Grassroots Supercomputing”, *Science*, vol. 308, pp. 810-813, 2005.

38 Condor staff, “What is Condor?”, http://www.cs.wisc.edu/condor/description.html, Accessed 2/22/05.

39 L. Smarr and C. Catlett, “Metacomputing”, *Communications of the ACM*, vol. 35, pp. 44-52, June, 1992.

40 D. Eager, E. Lazowska, and J. Zahorjan, “Adaptive Load Sharing in Homogeneous Distributed Systems”, *IEEE Transactions on Software Engineering*, vol. SE-12, num 5, pp. 662-675, May 1986..

41 R. van Nieuwpoort, T. Kielmann, & H. Bal, “Satin: Efficient Parallel Divide-and-Conquer in Java”, *Proc. Euro-PAR*, pp. 690-699, 2000, [Online] Available: http://www.cs.vu.nl/~rob/papers/europar2000.ps.gz.

42 J. Baldeschweiler, R. Blumofe, and E. Brewer, “ATLAS: An Infrastructure for Global Computing”, *Proc. Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996, [Online] Available: http://citeseer.ist.psu.edu/cache/papers/cs/577/http:zSzzSzmosquitonet.stanford.eduzSzsigops96zSzpaperszSzbaldeschwieler.pdf/baldeschwieler96atlas.pdf.

43 Z. Xu, “Simulation of Heterogeneous Networks of Workstations”, *Proc. MASCOTS*, pp. 59-63, 1996, [Online] Available: http://citeseer.ist.psu.edu/cache/papers/cs/759/http:zSzzSz128.105.7.11zSz~zhichenzSzpaperszSzmascots96.pdf/simulation-of-heterogeneous-networks.pdf.

44 Wikipedia (author unknown), “Supercomputer”, http://en.wikipedia.org/wiki/Supercomputer, Modified 8/17/06, Accessed 8/20/06.

45 The Economist, “A grid by any other name”, http://public.eu-egee.org/files/EconomistDec04.pdf. Modified 12/2/04, Accessed 6/24/06.

46 I. Foster, C. Kesselman, and S. Teucke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, *International J. Supercomputer Applications*, vol. 15, num. 3, 2001, [Online] Available: http://www.globus.org/alliance/publications/papers/anatomy.pdf.

47 I. Foster, “What is the Grid? A Three Point Checklist”, *GRIDtoday*, vol. 1, July, 2002, [Online] Available: http://www.gridtoday.com/02/0722/100136.html.

48 Sun Microsystems, Inc, “Sun Unveils Next Steps in its Vision of ‘The Network is the Computer’ Through Radical Simplification and a Call to ‘Share in the Power of the Grid’”, http://www.sun.com/smi/Press/sunflash/2005-02/sunflash.20050201.1.html, Accessed 4/29/05.

49 Srce, “CRO-GRID Infrastructure – Questions and Answers”, http://www.srce.hr/?id=599, Accessed 8/20/06.

50 MTA-SZTAKI LPDS, “P-GRADE Grid Portal”, http://www.lpds.sztaki.hu/pgportal/, Accessed 8/20/06.

51 The Greek Research and Technology Network “See-Grid: Overview”, http://www.see-grid.org/index.php?op=modload&modname=Sitemap&action= sitemapviewpage&pageid=2, Accessed 8/20/06.

52 J. Baldassari, “Design and Evaluation of a Public Resource Computing Framework”, M.S. Thesis, Worcester Polytechnic Institute, 2006.

53 Distributed.net staff, “distributed.net: Node Zero” http://www.distributed.net/, Modified 3/8/05, Accessed 4/1/05.

54 BOINC staff, “Preferences,” http://boinc.berkeley.edu/prefs.php, Modified 10/1/04, Accessed 2/24/05.

55 Distributed.net staff, “Distributed.net FAQ-O-Matic,” http://n0cgi.distributed.net/faq/cache/78.html, Accessed 2/24/05.

56 Grid.org staff, “GRID.ORG – Help: Frequently Asked Questions”, http://www.grid.org/help/faq_wus.htm, Accessed 4/1/05.

57 V. Pande, “Folding@home - FAQ-main”, http://folding.stanford.edu/English/FAQ-main, Modified 1/17/08, Accessed 9/19/08.

58 GIMPS staff, “Source Code”, http://www.mersenne.org/source.htm, Modified 8/9/05, Accessed 5/27/06.

59 BOINC staff, “Berkeley Open Infrastructure For Network Computing,” http://boinc.berkeley.edu/, Modified 4/6/06, Accessed 4/12/06.

60 V. Pande, “Folding@home Frequently Asked Questions”, http://folding.stanford.edu/faq.html, Modified 4/27/06, Accessed 5/27/06.

61 GIMPS staff, “Frequently Asked Questions”, http://www.mersenne.org/faq.htm. Modified 2/03, Accessed 5/27/06.

62 Seti@home staff, “Certificate of Computation”, http://setiathome.berkeley.edu/cert_print.php, Accessed 5/27/06.

63 D. Finkel, personal communication, 1/05.

64 D. Toth, “Volunteer Computing with Video Game Consoles” , *Proc. 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Computing and Systems*, February, 2007.

65 Sony Computer Entertainment, Inc. "Business Data" http://www.scei.co.jp/corporate/data/bizdataps2_e.html, Accessed 3/24/06.

66 Wikipedia (author unknown), "Atari 2600 – Wikipedia, the free encyclopedia" http://en.wikipedia.org/wiki/Atari_2600, Modified 3/25/06, Accessed 3/25/06.

67 S. Ewen and L. Lemarie, "Console Yourself", http://www.technology.scee.net/files/presentations/agdc2002/ConsoleYourself.pdf, Modified 2002, Accessed 3/01/06.

68 Wikipedia (author unknown), "PlayStation 2 – Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/PS2, Modified 3/25/06, Accessed 3/25/06.

69 Wikipedia (author unknown), "Xbox – Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/Xbox, Modified 3/25/06, Accessed 3/25/06.

70 Microsoft, "Xbox.com | XBOX 360 – The System", http://www.xbox.com/en-US/hardware/xbox360/default.htm, Accessed 3/25/06.

71 Seti@home staff, "Porting and optimizing SETI@home", http://setiweb.ssl.berkeley.edu/sah_porting.php, Accessed 10/1/05.

72 Seti@home staff, "Seti@home source code", http://setiweb.ssl.berkeley.edu/sah/seti_source/nightly/seti_boinc-client-cvs-2005-10-01.zip, Modified 10/1/05, Accessed 10/1/05.

73 E. Martinian, "Fast Fourier Transform C Code", http://www.csua.berkeley.edu/~emin/source_code/fft/, Accessed 9/30/05.

74 E. Martinian, source code, http://www.csua.berkeley.edu/~emin/source_code/fft/fft.c, Accessed 9/30/05.

75 V. Pandey, "PS3 FAQ," http://www.stanford.edu/group/pandegroup/folding/FAQ-PS3.html, Modified 3/26/07, Accessed 2/10/08.

76 D. Anderson, E. Korpela, and R. Walton, "High-Performance Task Distribution for Volunteer Computing", *Proc. First IEEE International Conference on e-Science and Grid Technologies*, 2005, [Online] Available: http://boinc.berkeley.edu/boinc_papers/server_perf/server_perf.pdf.

77 BOINC staff, "Workunits," http://boinc.berkeley.edu/work.php, Modified 11/25/04, Accessed 2/23/05.

78 BOINC staff, "Work Distribution", http://boinc.berkeley.edu/work_distribution.php, Modified 11/11/04, Accessed 2/23/05.

79 D. Toth, and D. Finkel, “A Comparison of Techniques for Distributing File-Based Tasks for Public-Resource Computing”, *Proc. 17th IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 398-403, November, 2005.

80 L. Cristofor, “ARMiner Info Index”, http://www.cs.umb.edu/~laur/ARMiner/, Modified 12/26/02, Accessed 4/9/05.

81 W. Richardson, D. Avondolio, J. Vitale, S. Schrager, M. Mitchell, and J. Scanlon, *Professional Java JDK 5 Edition*, Indianapolis, IN: Wily Publishing, Inc., 2005.

82 D. Toth and D. Finkel, “Characterizing Resource Availability for Volunteer Computing and its Impact on Task Distribution Methods”, *Proc. 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Computing and Systems*, February, 2007.

83 G. Woltman, personal communication, 9/14/05.

84 D. Anderson, personal communication, 8/16/05.

85 J. Brevik, D. Nurmi, and R. Wolski, “Automatic Methods for Predicting Machine Availability in Desktop Grids and Peer-to-peer Systems”, *Proc. IEEE Symposium on Cluster Computing and the Grid*, April 2004, [Online] Available: http://pompone.cs.ucsb.edu/~rich/publications/quant-est.pdf.

86 M. Mutka, and M. Livny, “Profiling Workstations’ Available Capacity For Remote Execution”, *Proc. 12th IFIP WG 7.3 International Symposium on Computer Performance Modeling, Measurement and Evaluation*, pp. 529-544, 1987, [Online] Available: www.cs.wisc.edu/condor/doc/profiling_availability.pdf.

87 A. Acharya, G. Edjlali, and J. Saltz, “The Utility of Exploiting Idle Workstations for Parallel Computation”, *Proc. SIGMETRICS'97*, 1997, [Online] Available: http://citeseer.ist.psu.edu/cache/papers/cs/1392/ftp:zSzzSzftp.cs.umd.eduzSzpubzSzpaperszSzpaperszSzncstrl.umcpzSzCS-TR-3710zSzCS-TR-3710.pdf/acharya97utility.pdf.

88 D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien, "Characterizing and Evaluating Desktop Grids: An Empirical Study", *Proc. of the International Parallel and Distributed Processing Symposium*, April 2004, [Online] Available: http://www.lri.fr/~dkondo/pubs/ipdps04_kondo.pdf.

89 D. Kondo, “Scheduling Task Parallel Applications For Rapid Turnaround on Desktop Grids”, Ph.D. Dissertation, University of California, San Diego, 2005, [Online] Available: http://www.lri.fr/~dkondo/pubs/thesis_kondo.pdf.

90 D. Kondo, G. Fedak, F. Cappello, A. Chien, and H. Casanova, “Resource Availability in Enterprise Desktop Grids”, *Technical Report 00000994*, INRIA, January, 2006.

91 S. Chong, “HOWTO track a user's idle time”, http://www.codeproject.com/dll/trackuseridle.asp, Modified 11/13/01, Accessed 7/20/05.

92 S. Chong, “Idle tracking source code”, http://www.codeproject.com/dll/TrackUserIdle/TrackUserIdle_src.zip, Accessed 7/20/05.

93 sfwong (user name, actual name unknown), “HOWTO track a user’s idle time”, http://www.codeproject.com/dll/trackuseridle.asp?df=100&forumid=2894#xx530130xx Modified 6/20/03, Accessed 7/20/05.

94 Rockwell Software, “Rockwell Automation – Arena Simulation Sofware”, http://www.arenasimulation.com/, Modified 2008, Accessed 2/9/08.

95 D. Kondo, “Desktop Grid Trace Archive”, http://xw01.lri.fr:4320/dg/, Accessed 2/12/08.

96 D. Toth and D. Finkel, “Increasing the Amount of Work Completed by Volunteer Computing Projects with Task Distribution Policies”, *Proc. of 2nd Workshop on Desktop Grids and Volunteer Computing Systems*, April, 2008.

97 Aurora Borealis (user name, actual name unknown), “SETI@home Beta – some questions”, http://setiathome.berkeley.edu/forum_thread.php?id=37561, Modified 2/5/07, Accessed 2/8/07.

98 Author unknown, “Result Deadline – Unofficial BOINC Wiki”, http://boinc-wiki.ath.cx/index.php?title=Deadline, Modified 2/2/06/, Accessed 2/9/07.

99 Keck_Computers (user name, actual name unknown), “Posts by Keck_Komputers”, http://einstein.phys.uwm.edu/forum_user_posts.php?userid=2914, Accessed 2/9/07.

100 V. Pandey, “Folding@Home News”, http://folding.stanford.edu/news.html, Modified 1/22/07, Accessed 2/9/07.

101 V. Pandey, “Folding@ Home configuration FAQ”, http://folding.stanford.edu/FAQ-settings.html, Modified 1/1/07Accessed 2/9/07.

102 V. Pandey, “Frequently Asked Questions (FAQ)”, http://folding.stanford.edu/faq.html, Accessed 2/9/07.

103 Author unknown, “WorkUnits – FaHWiki”, http://fahwiki.net/index.php/WorkUnits, Modified 1/24/07, Accessed 2/8/07.

104 Graham G3ZOD (user name, actual name unknown), “Workunit size vs. processor”, http://einstein.phys.uwm.edu/forum_thread.php?id=4583, Modified 7/25/06, Accessed 2/8/07

105 B. Allen, “Einstein@Home FAQ”, http://einstein.phys.uwm.edu/faq.php, Accessed 2/8/07.

106 Wikipedia (author unknown), “QMC@Home - Wikipedia, the free encyclopedia”, http://en.wikipedia.org/wiki/QMC%40Home, Accessed 2/8/07.

107 Author unknown, “Report deadline too short”, http://lhcathome.cern.ch/forum_thread.php?id=1977, Modified 1/23/06, Accessed 2/8/07.

108 Author unknown,“get wu’s with deadline less than 5 days, why??”, http://lhcathome.cern.ch/forum_thread.php?id=1619. Modified 8/30/05, Accessed 2/9/07.

109 Author unknown, “grid.org Forums – View topic – READ ME -=- Work Units (WU)”, http://forum.grid.org/phpBB/viewtopic.php?t=8847&highlight=workunit+size, Modified 1/14/06, Accessed 2/8/07.

110 Rosetta@home staff, “Rosetta@Home FAQ (work in progress), http://boinc.bakerlab.org/rosetta/forum_thread.php?id=669, Modified 6/10/06, Accessed 10/23/06.

111 ClimatePrediction.Net staff, “ClimatePrediction.Net gateway”, http://climateapps2.oucs.ox.ac.uk/cpdnboinc/quick_faq.php, Accessed 2/8/07.

112 SIMAP staff, “BOINCSIMAP :: View topic – Wus”, http://boinc.bio.wzw.tum.de/boincsimap/forum/viewtopic.php?t=5, Modified 11/29/05, Accessed 2/8/07.

113 tneural (user name, actual name unknown), “The Riesel Sieve Project :: View Topic – Length of WU”, http://www.rieselsieve.com/forum/viewtopic.php?t=819, Updated 8/20/06, Accessed 2/9/07.

114 L. Stephens and B. O’Shea, “PerlBOINC :: RieselSieve”, http://boinc.rieselsieve.com/?faq, Accessed 2/8/07.

115 B. (last name unknown), “The Riesel Sieve Project :: View topic – Checkpointing?”, http://www.rieselsieve.com/forum/viewtopic.php?t=1084, Accessed 2/9/07.

116 Viktors (user name, actual name unknown), "World Community Grid – View Thread – Run times for work units – what to expect", http://worldcommunitygrid.org/forums/wcg/viewthread?thread=928, Modified 12/10/04, Accessed 2/8/07.

117 J. Young, "A First Order Approximation to the Optimum Checkpoint Interval", *Communications of the ACM*, vol. 17, pp. 530-531, September 1974.

118 E. Gelenbe and D. Derochette, "Performance of Rollback Recovery Systems under Intermittent Failures", *Communications of the ACM*, vol. 21, pp. 493-499, June, 1978.

119 E. Gelenbe, "On the Optimum Checkpoint Interval", *Journal of the Association for Computing Machinery*, vol. 26, pp. 259-270, April 1979.

120 N. Vaidya, "Impact of Checkpoint Latency on Overhead Ratio of a Checkpointing Scheme", *IEEE Transactions on Computers*, vol. 46, pp. 942-947, August 1997.

121 J. Plank and M. Thomason, "The Average Availability of Uniprocessor Checkpointing Systems, Revisited", *Technical Report UT-CS-98-400*, Department of Computer Science, University of Tennessee, August 25, 1998, [Online] Available:

122 "BOINCstats – BOINC Statistic / BAM! – BOINC Account Manager", http://boincstats.com/, Modified 2/3/08, Accessed 2/4/08.

123 N. Cardwell, S. Savage, and T. Anderson, Modeling TCP Latency, *Proceedings of IEEE INFOCOM 2000*, pp. 1742-1751. , March 2000, [Online] Available: http://www.cs.ucsd.edu/~savage/papers/Infocom2000tcp.pdf.

124 J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 303-314, August 1998, [Online] Available: http://www.sigcomm.org/sigcomm98/tp/paper25.pdf

125 J. Davies and T. Lee, *Microsoft Windows Server 2003 TCP/IP Protocols and Services Technical Reference*, 2003.

126 T. White, "Scheduling recurring tasks in Java applications," http://www.ibm.com/developerworks/java/library/j-schedule.html, Modified 11/4/03, Accessed 2/12/08.