

# A COMPARISON OF TECHNIQUES FOR DISTRIBUTING FILE-BASED TASKS FOR PUBLIC-RESOURCE COMPUTING

David Toth and David Finkel  
Department of Computer Science  
Worcester Polytechnic institute  
100 Institute Road  
Worcester, MA 01609-2280  
U.S.A.  
toth@cs.wpi.edu, dfinkel@cs.wpi.edu

## ABSTRACT

Public-resource computing (PRC) projects use volunteered computational resources in order to accomplish some goal [1]. Because the projects are so computationally intensive and only a small percentage of the public participates in them, improvements in efficiency are very important. One way to improve the efficiency of PRC projects is to decrease the amount of time wasted duplicating work unnecessarily. One cause of this waste is distributing tasks to clients in a sub-optimal manner. This work focuses on file-based tasks that require clients to download data files to perform the tasks. We propose a method of distributing tasks and compare the waste it produces to the waste created by the other currently used techniques.

## KEY WORDS

Distributed Computing, Public-Resource Computing, Performance

## 1. Introduction

Public-resource computing (PRC) projects use volunteered computational resources in order to accomplish some goal [1]. These projects allow people to solve many problems that were previously computationally infeasible. The power of public-resource 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. A handful of PRC projects are currently running, including the well-known SETI@home project, which searches for evidence of extraterrestrial life [2].

A public-resource 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 servers distribute the tasks 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, it spends the time working on tasks that a server assigns it to perform. In order to work on a task, the client must first download a file from the server. When the client has finished a task, it returns the result. 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 it was interrupted.

PRC projects distribute tasks using one of several techniques. Some PRC projects have their servers distribute only one task to a client at a time. When the client completes the task and returns the result to the server, the server assigns another task to the client. Other PRC projects have their servers send multiple tasks to a client when the client needs work. In this case, a client downloads any necessary files for each task and works on one task, storing the others in a buffer until the client has finished the task it is processing. Several PRC projects allow the user to configure how much work the client will buffer by allowing the user to specify the number of hours of work to buffer. These projects use benchmarks and information about the computer on which the client is executing to estimate how long it will take the computer to complete each task. When the client downloads work to fill its buffer, it downloads some whole number of tasks such that the estimated time to complete the tasks approximates the amount of hours of work to buffer. In some cases, a client may need to complete a task by a certain time. In these cases, the server will inform the client to abort a task that is late, and the client will begin processing the next task it has.

Because statistics show that very few people participate in PRC projects, it is crucial that PRC projects utilize the donated CPU time as efficiently as possible in order to maximize the amount of information they collect. Fewer than 1% of an estimated 300 million internet-connected personal computers participate in PRC projects [3]. Efficiently using the donated CPU time involves multiple challenges, some of which are at odds with one

another. These challenges include ensuring clients always have work, ensuring clients do not duplicate work unnecessarily, and making the donated cycles run the fastest code possible. In this study, we focus on minimizing duplicated work caused by the method of distributing tasks.

## 2. Related Work – Task Distribution in PRC Projects

Past and current PRC projects have used various methods of distributing tasks to clients. Currently there are several major file-based PRC projects running from different organizations. Active projects include several projects that utilize the Berkeley Open Infrastructure for Network Computing (BOINC) [2], two projects from Distributed.net [4], and two projects from Grid.org [5]. In addition to these projects, another project framework, Xtremweb, is available for use [6]. The Satin [7] and ATLAS [8] projects also explored distributing work.

The projects that use the BOINC framework call each task a workunit. Each workunit has an associated estimation of the number of floating point operations that it requires to be completed, as well as an associated amount of memory and disk space that are required [9,10]. A BOINC project will not send a workunit to a client that does not have the required amount of memory or free disk space [9,10]. A project also does not send a workunit to a client if the client's CPU speed and usage parameters indicate the work will not be completed in some acceptable amount of time [9,10]. 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 [11]. Thus, while BOINC uses benchmarking for determining which tasks to send to a client, it allows the user to determine how much data to send to the client. This allows 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 [12]. Using a quick benchmark, the project estimates how much data a client can process in a given amount of time [12]. As with BOINC, this method does not send each client a task that is the optimal size, but instead, sends the user an amount of data that the user requests. This method 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 distributing tasks that both BOINC and Distributed.net use as the BOINC method.

Conversely, Grid.org and Xtremweb both distribute tasks one 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 [13,14]. Neither Grid.org nor Xtremweb claim to take into account the abilities of the computer where a client is running when they

distribute tasks. Thus, a client may receive a task that it will not complete in a reasonable amount of time. For convenience, we refer to the method of distributing tasks that both Grid.org and Xtremweb use as the Grid.org method.

Another method of distributing work to clients was explored in systems such as Satin and ATLAS. These systems used work stealing to distribute tasks [7,8]. In this process, 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 [7,8]. Work stealing performs some load balancing of the system automatically [7]. However, a consequence of using work stealing is that tasks are sent over the network a second time when they are stolen.

## 3. Comparison of Task Distribution Methods

We wish to investigate the relative strengths of different methods of distributing tasks to clients. 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 BOINC, has clients download multiple tasks at one time and buffer them. The method used by Grid.org and Xtremweb, which we call Grid.org, 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. 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, whenever the client finishes a task, there is another one waiting for it. Both the *Buffer1* and BOINC 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 distributing tasks wastes.

## 4. Comparing Wasted Time for the Systems

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

We analyzed the different methods that the various projects use to distribute tasks to clients 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,

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

We note that for BOINC based projects, Distributed.net projects, and projects using the *Buffer1* method,  $W = 0$  because at least one task is always downloaded before the last one in the buffer is completed. 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 many computers are fast enough to be able to perform other instructions while downloading a file.

## 5. Simulations

### 5.1 Simulation Design

Using Equation 1 from our model in Section 4, we developed a simulation to gather data about the amount of wasted time incurred by the different methods of distributing 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 frequency and duration of the interruptions that the client experienced. During the interruptions, 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.
- Buffered Hours. The number of hours of work a BOINC client buffers, not relevant for the *Buffer1* and Grid.org methods). If the number of

hours of work to buffer was not an even 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  $\div$  time to complete a task). We consider the BOINC method with the various amounts of hours buffered as different methods for our measurements, and we call the method BOINC- $x$  where  $x$  is the number of hours buffered.

- Distribution Method. The method of distributing tasks to clients.
- 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.
- Interruption Frequency. The time between the end of an interruption and the beginning of the next interruption (in hours).
- Interruption Duration. The duration of the interruptions (in hours).

We also ran the simulation with additional values of the abort multiple (2, 6), the mean frequency of interruptions (4 and 16 hours), and the mean duration of interruptions (2 and 8 hours) to gain additional insight as to how those variables would affect the amount of time the distribution methods would waste.

### 5.2 Simulation Development

We wrote a program in Java to perform the simulations. The simulations were run for 150 replications with each combination of the simulation parameters specified in Section 5.1. Each replication simulated a PRC client for a period of 2000 hours where time is divided into units of 1 second. We used a library from the ARMiner project to generate the random numbers we needed for the exponential distribution of interruption times and durations [15]. For each combination of simulation parameters, the random number generators were seeded with the same seed to ensure consistency between parameter combinations. Thus, the  $i^{\text{th}}$  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.

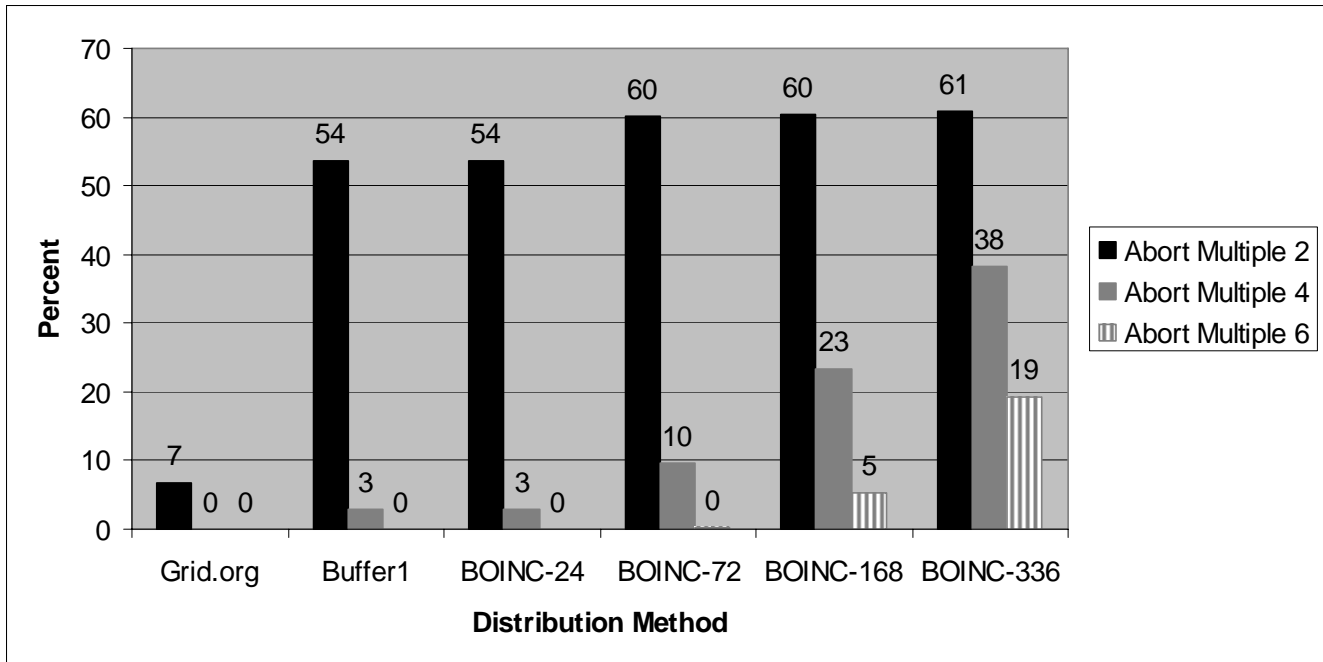
## 6. Results

We ran our simulation using the download speed of 300,000 bps, varying the abort multiple and the task distribution method, as those are the factors that the creators of PRC projects can control, and thus knowing the effects they have on waste should be useful to project

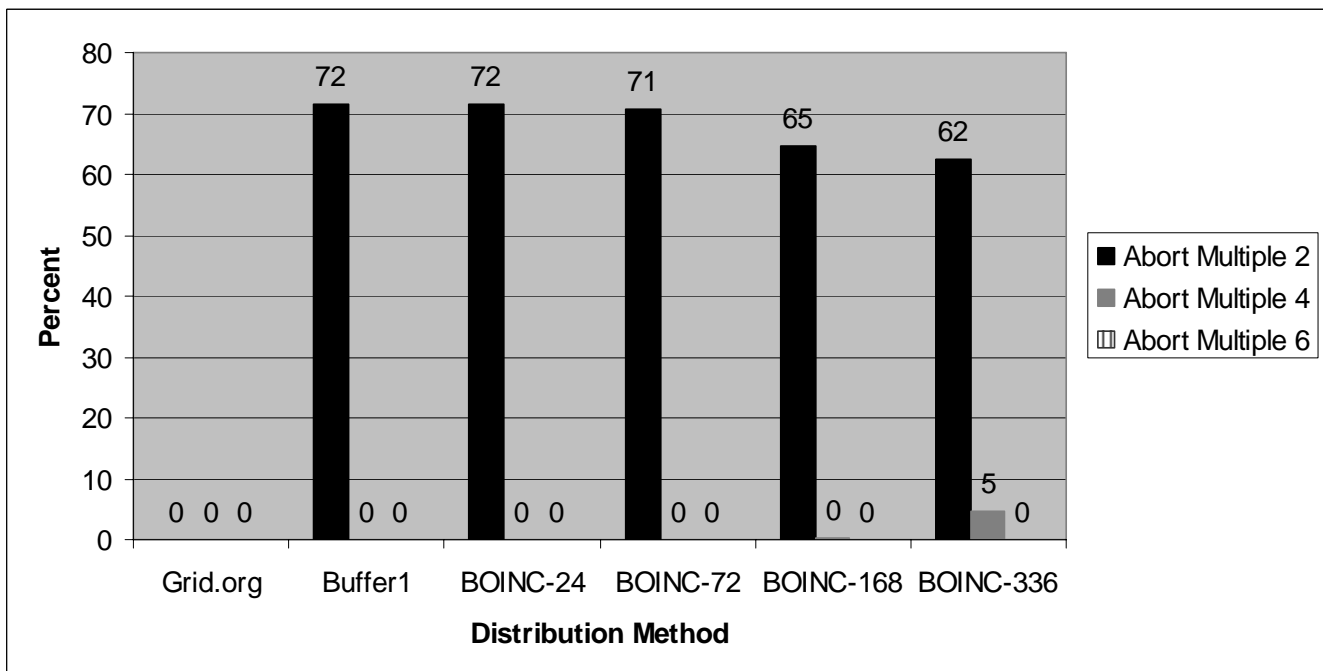
developers. The first set of simulations assumed an interruption frequency of 8 hours and an interruption duration of 4 hours. Figure 1 shows the percent of time wasted under those conditions. The Grid.org method wasted significantly less time than the other methods with the abort multiple 2. When the abort multiple was 4, Grid.org still wasted far less time than BOINC-72,

BOINC-168, and BOINC-336, but only a little less than Buffer1 and BOINC-24. BOINC-24 wasted the same amount of time as Buffer1 for abort multiples 2 and 4. However, when the abort multiple was increased to 6, Buffer1, BOINC-24, and BOINC-72 were as efficient as Grid.org. Decreasing the abort multiple decreased the amount of wasted time for all the methods of distributing

**Figure 1 - Time Wasted For Interruption Frequency 8 and Interruption Duration 4**



**Figure 2 - Time Wasted For Interruption Frequency 16 and Interruption Duration 2**



tasks, and as the number of hours buffered by a BOINC 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 interruption frequency was 16 hours and the interruption duration was 2 hours. Figure 2 shows the results of the simulation with the abort multiples 2, 4, and 6, the interruption frequency 16, and the interruption duration 2; for several of these parameter settings, the wasted time was 0. Although when the abort multiple was 2, the Grid.org method was significantly better than the other methods, when the abort multiple was 4 or 6, there was almost no difference between the amount of time wasted by the different methods. With the exception of BOINC-168 and BOINC-336 for the abort multiple 4, 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 distribution methods to waste about the same amount of time. We tested file sizes of 1, 5, and 10 MB and download speeds of 56000, 300000, 1000000, and 10000000 bps. The different values of both parameters had almost no impact on the simulation results. We tested the values 1, 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 distribution methods wasted for 1 hour tasks. However, the Grid.org method wasted far less time for the other task completion times. We also ran simulations with abort multiples of 1 and 8, which resulted in almost identical amounts of wasted time for the different methods of distributing tasks.

## 7. Conclusions

The results of our simulation have shown two important conclusions. The results showed that sending only one task to a client instead of sending multiple tasks to a client at one time 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 distribution methods, the effect of a low abort multiple on the Grid.org method causes far less wasted time than it does on the other task distribution methods. Based on our results, we encourage developers of PRC projects to adopt the technique of sending only one task at a time to each client. We also suggest that the developers ensure that the tasks that a server sends to a

client will be able to be completed quickly enough that few tasks will need to be aborted. The adoption of these two practices should decrease the waste of PRC clients, thus enabling the clients to do more work in the same amount of time. However, we note that for people who contribute to PRC projects using a slow internet connection, it might be desirable to have the clients buffer tasks and assign less time critical tasks. This would minimize the idle cycles that cannot be used because the client has finished its task but cannot get a new one because the computer is not connected to the internet, thus making the BOINC distribution method a good idea for those users.

## 8. Future Work

In future work, we will explore the effects of interruptions halting tasks instead of pausing them, requiring the tasks to be restarted from the beginning. We will incorporate checkpointing into our simulations and attempt to determine if restarting the tasks and the use of checkpointing affects which method of distributing tasks causes the least amount of wasted time. It does not appear that any public-resource computing projects have collected the actual data on frequency and duration of interruptions. We are working on a project to collect this data from several different groups of users. Once we have collected the data, we will incorporate it into our simulations. We will further refine our simulations by incorporating any available information about the values of the other parameters that are used in actual PRC projects.

## References:

- [1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, & D. Werthimer, SETI@home: An Experiment in Public-Resource Computing, *Communications of the ACM*, 45(11), 2002, 56-61.
- [2] "Berkeley Open Infrastructure For Network Computing," <http://boinc.berkeley.edu/>, Updated 2/1/05, Accessed 2/9/05.
- [3] J. Bohannon, Grassroots Supercomputing. *Science* 308, 2005, 810-813.
- [4] Distributed.net, "distributed.net: Node Zero" <http://www.distributed.net/>, Updated 3/8/05, Accessed 4/1/05.
- [5] Grid.org, "GRID.ORG – Project Overview", <http://www.grid.org/projects/>, Accessed 4/1/05.

- [6] "XtremWeb," <http://www.lri.fr/~fedak/XtremWeb/introduction.php3> Updated 2/8/05, Accessed 2/19/05.
- [7] R. van Nieuwpoort, T. Kielmann, & H. Bal, Satin: Efficient Parallel Divide-and-Conquer in Java. *Proc. Euro-PAR*, Munich, Germany, 2000, 690-699.
- [8] J. Baldeschweiler, R. Blumofe, & E. Brewer, ATLAS: An Infrastructure for Global Computing. *Proc. Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, Connemara, Ireland 1996.
- [9] "Workunits," <http://boinc.berkeley.edu/work.php> Updated 11/25/04, Accessed 2/23/05.
- [10] "Work Distribution," [http://boinc.berkeley.edu/work\\_distribution.php](http://boinc.berkeley.edu/work_distribution.php) Updated 11/11/04, Accessed 2/23/05.
- [11] "Preferences," <http://boinc.berkeley.edu/prefs.php> Updated 10/1/04, Accessed 2/24/05.
- [12] Distributed.net, "Distributed.net FAQ-O-Matic," <http://n0cgi.distributed.net/faq/cache/78.html>, Accessed 2/24/05.
- [13] G. Fedak, C. Germain, V. Neri, & F. Cappello, XtremWeb: A Generic Global Computing System. *Proc. 1<sup>st</sup> International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, 2001, 212-220.
- [14] Grid.org, "GRID.ORG – Help: Frequently Asked Questions", [http://www.grid.org/help/faq\\_wus.htm](http://www.grid.org/help/faq_wus.htm), Accessed 4/1/05.
- [15] Laurentiu Cristofor, "ARMiner Info Index", <http://www.cs.umb.edu/~laur/ARMiner/>, Updated 12/26/02, Accessed 4/9/05.