

Increasing the Amount of Work Completed by Volunteer Computing Projects with Task Distribution Policies

David Toth and David Finkel
Worcester Polytechnic Institute
Computer Science Department
Worcester, MA 01609
USA
toth@cs.wpi.edu, dfinkel@cs.wpi.edu

Abstract

Volunteer computing projects rely on volunteers running clients on their computers that contribute to projects when the computers' owners allow them to. These projects allow people to solve problems that were previously too computationally intensive to solve. However, due to the relatively small fraction of the population that participates in volunteer computing projects, it's very important to use the donated CPU cycles as efficiently as possible. Volunteer computing clients use two different methods to retrieve tasks: retrieving one task at a time when the client has no more work to do or retrieving multiple tasks at once and storing them in a buffer. We simulate these different task retrieval policies to compare the number of tasks completed by clients using the different policies. Our simulations showed that clients that retrieve one task at a time complete more tasks than clients that retrieve multiple tasks at once and buffer them. Our simulations also showed that there was not a significant gain in the amount of work that could be completed by devising a more complicated adaptive policy.

1. Introduction

Volunteer computing projects are often used for scientific, mathematical, and medical research, such as trying to find cures for diseases like cancer and AIDS. A project uses servers provided by the project's sponsor and computers volunteered by people. The servers distribute work to the volunteered computers and aggregate the results sent back by the volunteered computers. Because the volunteered computers do the work for the project, a volunteer computing project relies on

people to decide to participate and download and install a client program on their computer. The client typically runs either as the computer's screensaver or as a service/daemon process in the background. When the client is running, it obtains a task or multiple tasks to do for a project, downloads any files required to perform the tasks, and returns the results of the tasks to the servers as it completes them.

Volunteer computing projects have become an effective way to make progress on problems that would be otherwise computationally infeasible. Over the last couple of years, the number of volunteer computing projects has increased drastically, expanding from the well-known SETI@home, Folding@home, and the Great Internet Mersenne Prime Search (GIMPS) projects, to many newer projects like Einstein@home and LHC@home [1, 2, 3, 4, 5]. The increase is likely due to the development of the Berkeley Open Infrastructure for Network Computing (BOINC) framework, which allows people to create volunteer computing projects much quicker and easier than having to write them from scratch every time [6].

Unfortunately, although the number of projects has increased, only a small portion of the Internet-connected computers in the world participate in volunteer computing. It was reported that of the estimated 300 million Internet-connected personal computers, less than 1% participate in volunteer computing projects [7]. Since the projects are so computationally intensive, it is important to attract more volunteers and also to use the CPU cycles that people currently volunteer more effectively. One aspect of volunteer computing project software that could be changed is the task retrieval policy. Changing the policy could enable the volunteered CPU cycles to be used more effectively.

In some volunteer computing projects, the

results of some tasks are used to create new tasks or as input to other tasks, making it important to get the results of the tasks back in a timely manner [8]. It is also important to get the results of tasks back quickly to finish the projects as soon as possible. Although assigning the same task to multiple clients at the same time might get a task done faster, the duplication of effort also wastes volunteered CPU time. In an attempt to ensure that the results of tasks get returned as quickly as possible, many volunteer computing projects have deadlines associated with each of the tasks the clients perform. If the result of a task is not returned by the task's deadline, a server will reassign the task to another client and the client that was working on the task aborts the task.

Volunteer computing clients request tasks to process using one of two techniques. The first technique has clients request multiple tasks at a time and store them in a buffer [9, 10]. The client processes one task at a time, returning the result to the server, and then beginning to process the next task in the buffer. When the buffer has few enough tasks left in it, the client requests more from the server. The other method that is used is a client will request one task, complete it, return the result, and then request another task [8, 11, 12]. Both of these methods are in use today by different projects. Because more than one method is currently used to retrieve tasks, we suspected that this was an area of volunteer computing that might be able to benefit from an analysis of the task retrieval techniques. In previous work, we explored the area of different task retrieval policies and determined there was potentially a difference in the work a client might accomplish using the different task distribution methods [13].

2. Methodology

In order to determine whether clients would complete more tasks using one of the task retrieval methods, we first had to be able to accurately simulate a number of volunteer computing clients and thus, we needed to construct a volunteer computing client simulator. Making the simulator mimic a real volunteer computing client required a great deal of information. We needed information about:

1. When computers running the clients would be off, on, and running their screensavers.
2. How the task retrieval policies are implemented, including how the policy where clients download multiple tasks at a

time and buffer them was implemented in volunteer computing projects.

3. How checkpointing works in volunteer computing projects.
4. Which parameters would play a key role in volunteer computing projects and typical values of those parameters.

2.1 Computer Trace Data

As we discussed in the list in Section 2, we needed information about when computers running the clients would be off, on, and running their screensavers. We were unable to use data collected by others in the past because that data did not match our sampling interval and the number and variety of computers about which we needed data. Thus, we collected traces of computer usage to use in the simulations [14]. The traces of the computer usage were from 140 computers, consisting of home computers, computers belonging to undergraduate students at Worcester Polytechnic Institute, computers in public labs at Worcester Polytechnic Institute, and computers from a business that requested to remain anonymous. A Windows service was used to collect the traces. The service recorded the status of the computer's screensaver every 10 seconds for a period of 28 days. Using the data, we were able to deduce when the computer was powered off, too.

2.2 Task Retrieval Policies

We simulated three different task retrieval policies in our work which are summarized in Table 1. The policies we chose include the two currently used by existing volunteer computing projects: (1) having the client buffer some number of day's worth of tasks (*Buffer Multiple Tasks*) and (2) having the client buffer no tasks at all and retrieve the next task only after returning the result to the one task it was working on (*Buffer None*). For the *Buffer Multiple Tasks* policy, we had the client download 1 week of tasks in one run of the simulations and 2 weeks of tasks in another run of the simulations. We also simulated (3) the *Super-Optimal* task retrieval policy, a theoretical policy that provides an upper bound on the number of tasks that could be completed by a client.

The first task retrieval policy we simulated is the *Buffer Multiple Tasks* policy which is the policy used by the BOINC framework [15]. Because the BOINC framework is used to create the vast majority of the existing volunteer computing projects, this policy is extremely important. The

Table 1 - Task retrieval policies

	Policy Name	Description
1	Buffer Multiple Tasks	Buffers some number of days of tasks. The amount of tasks is limited to a number that can possibly be completed before the tasks' deadlines.
2	Buffer None	Does not buffer any tasks. Downloads a task after returning the result of the previous task.
3	Super-Optimal	Does not buffer any tasks. Downloads a task after returning the result of the previous task, but assumes tasks are downloaded instantaneously. Does not attempt to execute a task that will not be completed on time.

BOINC framework allows users to configure their client, specifying how much work they want the client to download at one time [9]. However, BOINC will override the user's settings by limiting the client to downloading only tasks that it determines the client might be able to complete on time [16]. This limit makes the policy significantly better than it would otherwise be, by reducing the number of uncompleted tasks and thereby decreasing wasted time spent on tasks that wouldn't be completed on time. In turn, this makes the CPU time spent on tasks more likely to be spent on tasks that will be completed on time, increasing the number of tasks a client will complete in a given amount of time. To simulate the *Buffer Multiple Tasks* policy (BOINC's policy), when a client requests work, the simulator first determines how much CPU time the client is expected to receive before a new task would need to be completed. The expected amount of CPU time is calculated by multiplying the percentage of time the computer was available during the 28 day trace by the wall clock time before the new task would be aborted. The simulator divides the expected amount of CPU time by the time it should take to complete a task, getting the maximum number of tasks that are expected to be completed before the new task would be aborted. If the client is buffering fewer tasks than the maximum number of tasks that could be completed before the new task would be aborted, then the new tasks are retrieved. Otherwise, the new tasks are not retrieved.

The *Buffer None* policy, which is used by Grid.org's volunteer computing projects, ensures that a client has the most possible wall clock time to process a task since there are no other tasks that might be processed during that time like there are with the *Buffer Multiple Tasks* policy [12]. However, the CPU of the computer where the client is running is likely to have spare cycles while the next task is being downloaded, since downloading a file primarily involves the network card, the system

bus, and the hard disk and requires a minimal amount of work by the CPU. Therefore, the CPU time during the process of downloading a new task is almost completely wasted. The wasted time can become significant in the case where a task requires a large data file and the computer has a relatively slow Internet connection, particularly if the tasks are short.

The *Super-Optimal* policy is similar to the *Buffer None* policy in that it doesn't have the client download a new task until it has returned the task it is working on. However, in the *Super-Optimal* policy, the act of downloading the next task is assumed to occur instantaneously, while the *Buffer None* policy is unable to work on the task until sufficient time to download the task has passed. Not downloading a task until it can be started minimizes the chance that the task is not completed on time by giving the computer the most wall clock time to complete the task before it would be aborted. Having the download take no time mimics the idea that downloading the task uses very little CPU time and mostly other resources such as the network card, which might otherwise be unused as the current task is being processed. An aspect of this policy that is even more significant is the look-ahead feature it uses. When the simulated client using the *Super-Optimal* policy is ready to download and start a task, it scans through the trace and determines if it would be able to complete the task on time. If the client is able to complete the task on time, it downloads the task and begins working on it. Otherwise, the client sleeps for one second and then checks to see whether it could complete a task if it downloaded and started it then. This method ensures that the most tasks will be completed by a client using the *Super-Optimal* policy by ensuring that a task that will not be completed does not use CPU time that could otherwise be used to complete a task. We note that the *Super-Optimal* policy clearly cannot actually be implemented in a volunteer computing client

because it requires knowledge of the computer's future availability. However, the *Super-Optimal* policy provides an upper bound on the number of tasks a client can complete in a given amount of time. The upper bound serves as a benchmark, allowing us to determine how effective other policies are and whether there is sufficient motivation to try to develop better policies.

2.3 Checkpointing

Volunteer computing clients implement checkpointing at regular intervals to avoid needing to restart any incomplete task from the beginning in event that the client is terminated for any reason, such as the computer being turned off. Our simulator incorporates checkpointing to accurately mimic volunteer computing clients. However, the data on how frequently checkpoints are created and how long it takes to create a checkpoint and restore the state of a task from a checkpoint was not available, and we believe that this likely varies from project to project. Therefore, we assume that volunteer computing projects checkpoint at the optimal interval to minimize the time it takes to complete the project. We estimated that it takes 10 seconds to create a checkpoint or to restore the state of a task from a checkpoint. We calculated the frequency that tasks would be interrupted based on the computer usage traces we collected. Using the estimate of time to create a checkpoint and restore from a checkpoint and the failure rate of tasks, we used Young's formula to calculate the optimal checkpoint interval for use in our simulations [17].

2.4 Simulation Parameters

We wanted to ensure that our simulations accurately showed which task retrieval policies enabled clients to complete the most tasks over a period of time under realistic conditions. Ideally, we would observe that a particular policy consistently allowed clients to complete more tasks. If we observed that different conditions produced different results, then volunteer computing project sponsors could select the policy that would work best for their project, based on the expected values of their project's parameters such as delay bound and file size. The parameters we used in the simulations were:

- **File Size.** The size of the file required by each task. The file is downloaded before the task can be started.

- **Download Speed.** The download speed of the client.
- **Completion Time.** The amount of CPU time required to complete a task if it is not interrupted.
- **Buffered Days of Work.** The number of days of work a client using the *Buffer Multiple Tasks* policy tries to buffer. This parameter is not relevant for the other task retrieval policies.
- **Retrieval Method.** The method clients use to retrieve tasks.
- **Delay Bound.** The amount of wall clock time between completing the download of a task and when the task will be aborted.
- **Checkpoint Time.** The time to create or restore a checkpoint.
- **Internet Connectivity.** We assume that the client is always connected to the Internet because we do not expect mobile devices to be used to run volunteer computing clients.

We gathered as much information as we could find about what parameter values specific volunteer computing projects use (shown in Table 2). This was an important step because we are unaware of any such summary of volunteer computing project parameter summaries from other work. Due to the lack of an existing summary, this information comes from a great deal of sources. Using the information we gathered on existing volunteer computing projects and calculations we performed to estimate the optimal checkpoint interval for volunteer computing projects using the formula from Young [17], we determined the parameters we used for our simulations. We adopted the following base parameter settings based on the data in Table 2:

- **File Size:** 1 MB
- **Download Speed:** 300 kbps, 10 mbps
- **Delay Bound:** 7 days
- **Checkpoint Cost and Frequency:** 10 seconds and 1574 seconds
- **Task Duration:** 4 hours, 24 hours

3. Results and Analysis

We ran the simulations for the task retrieval policies and the selected parameters using the traces we had obtained. We compared the total number of tasks each policy completed using all 140 traces and compared that to the number of tasks the

Table 2 - Parameter values for various volunteer computing projects

Project	File Size	Delay Bound	Task Duration	Checkpointing Frequency
SETI@home	350 k [18]	4 to 60 days [19]		10 seconds [20]
Folding@home	normal < 5 MB, some new large ones ~5 MB [21, 22]	max(10 days or 2 + 30*days to do on dedicated P4-2.8 machine) [8]	2-4 days on benchmark machine (P4-2.8) [23]	15 minutes [8]
Einstein@home	4.5 MB, 12 MB, 16 MB [24, 25]	1 week [25]		
QMC@home			4-48 hours [26]	
LHC@home		8 days, then 7 days, then 5 days [27, 28]		
Rosetta@home	800k, 1 MB, 1.2 MB, 1.5 MB, 3 MB [29]	(672 hours) changed to 168 hours [29]	3 hours, up to 24 hours, 2-4 days [29, 30]	varies based on user settings [30]
grid.org	10k - 100k [29]	100 hours, 150 hours, 200 hours, 222 hours, 336 hours [29]	20 hours on PII-400 [29]	
Climateprediction.net		347 days 5 hours 20 minutes or 150 days 5 hours 20 minutes [19]	3 weeks, 8 weeks, 20 weeks [31]	15-30 minutes [20]
SIMAP	1-2 MB [32]	10 days [19]	2 hours [32]	
The Riesel Sieve Project		7 days [33]	30 minutes [34]	10 minutes [35]
World Community Grid			10-20 hours [36]	

Super-Optimal policy completed. The results of the simulations are shown in Table 3. The percentages shown for the policies indicate how many fewer tasks the policies completed than the number of tasks the *Super-Optimal* policy completed. Thus, volunteer computing projects will complete more tasks by using a policy with a lower percentage. The clients completed more tasks when they used the *Buffer None* policy than when they used the *Buffer Multiple* policy. For our base parameter settings, the total number of tasks completed by the clients was greater for the *Buffer None* policy than for the *Buffer Multiple* policy when either one week or two weeks of tasks were buffered. For four-hour tasks, there is not a large difference in the number of tasks completed by the different policies. However, the difference between buffering one or two weeks of tasks as compared to not buffering any tasks is more significant when the tasks require

24 hours of CPU time to be completed. Based on these results, the *Buffer None* policy appears to be a better policy for retrieving tasks in volunteer computing projects for the given parameter settings than the *Buffer Multiple* policy where one or two weeks of tasks are buffered. Another point worth noting is that there is very little difference between the amounts of tasks that the clients complete when they use the *Buffer None* policy than the number of tasks that clients complete when they use the *Super-Optimal* policy. Therefore, it does not appear to make sense to try to devise an adaptive policy to try to dynamically determine how much work to buffer based on recent computer usage patterns, in an attempt to increase the number of tasks that could be completed by volunteer computing clients. Such a policy would not be able to increase productivity significantly.

In our simulations, we observed that buffering

tasks led to completing fewer tasks. This is a result of the delay bound for tasks. As discussed in Section 2.4, the delay bound is the amount of wall clock time between when the file required for a task is downloaded and when the task must be completed. Clients using the *Buffer None* policy may use all the available CPU cycles between when it is downloaded and when its delay bound is over. However, when a client uses the *Buffer Multiple Tasks* policy, the downloads for all of the files for the tasks assigned at one time complete at approximately the same time. Thus, the deadline

for those tasks is almost identical and all of those tasks must be completed in the available CPU cycles between when the files required for the tasks are downloaded and when the delay bound is over. In addition to all of those tasks needing to be completed, the last task from the previous set of tasks that was downloaded must be completed during that same block of available CPU cycles. Thus, if there are x usable CPU cycles between when the files for the tasks are downloaded and when the delay bound is over and if n tasks are downloaded at one time by a client using the Buffer

Table 3 – Differences between amount of tasks completed by *super-optimal* policy and other policies

Task Completion Time	Download Speed	Buffer None	Buffer 7 Days	Buffer 14 Days
4 hours	300 kbps	0.22%	0.94%	0.94%
4 hours	10 mbps	0.03%	0.93%	0.93%
24 hours	300 kbps	0.43%	3.33%	3.33%
24 hours	10 mbps	0.38%	3.33%	3.33%

Multiple Tasks policy, $n+1$ tasks must be completed in the x CPU cycles. In contrast to that, a client using the *Buffer None* policy has all x CPU cycles to complete one task. This leads to clients using the *Buffer Multiple Tasks* policy not completing tasks more often than clients using the *Buffer None* policy. If a task is not completed on time, all of the CPU time spent on that task is wasted. Over the course of our simulations of the 28-day traces, some of the time wasted by clients using the *Buffer Multiple Tasks* policy was used productively by clients using the *Buffer None* policy, allowing the clients using the *Buffer None* policy to complete as many as 3 tasks more than clients using the *Buffer Multiple Tasks* policy.

4. Conclusions and Future Work

The results of our simulations have shown two important conclusions. The first result of the simulations is they have shown that under realistic conditions, buffering tasks can lead to clients completing fewer tasks. The additional tasks completed by clients if the *Buffer None* policy is used in place of the *Buffer Multiple Tasks* policy should lead to significantly more tasks being completed when spread across the number of computers participating in volunteer computing

projects because the majority of the volunteer computing projects use BOINC’s *Buffer Multiple Tasks* policy.

The second result of the simulations is that they have shown that spending time devising an adaptive policy for task retrieval will likely yield minimal gains over the *Buffer None* policy. Our simulations have shown that the difference between the number of tasks completed by computers using the *Super-Optimal* policy and computers using the *Buffer None* policy is very small. Because this difference is so small, the gains that could be achieved by devising a more complex policy are minimal.

In the future, we plan to analyze the tradeoffs between using clients that run as screensavers, services (background processes running continuously), and web-based clients. We also note that our simulations have been based on single CPU computers. Because the trend today is to put multiple CPUs in home computers, we may adapt our simulations in the future to take into account that there may be multiple CPUs in computers that participate in volunteer computing. We also intend to study the effects of buffering tasks if the tasks take less CPU time to complete and if clients are not always connected to the Internet.

5. 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] Folding@Home. "Folding@Home Distributed Computing." <http://folding.stanford.edu/> Updated 2006. Accessed 10/26/06.
- [3] GIMPS. "Mersenne Prime Search". <http://www.mersenne.org/prime.htm> Accessed 7/13/05.
- [4] Einstein@Home. <http://einstein.phys.uwm.edu/> Updated 8/14/06. Accessed 10/26/06.
- [5] LHC@Home. <http://lhathome.cern.ch/> Updated 9/29/06. Accessed 10/26/06.
- [6] D. P. Anderson. BOINC: A System for Public-Resource Computing and Storage. 5th IEEE/ACM International Workshop on Grid Computing, November 8, 2004, Pittsburgh, USA.
- [7] J. Bohannon, Grassroots Supercomputing. *Science* 308, 2005, 810-813.
- [8] "Frequently Asked Questions (FAQ)", <http://folding.stanford.edu/faq.html>, Accessed 2/9/07.
- [9] "Preferences," <http://boinc.berkeley.edu/prefs.php> Updated 10/1/04, Accessed 2/24/05.
- [10] Distributed.net, "Distributed.net FAQ-O-Matic," <http://n0cgi.distributed.net/faq/cache/78.html>, Accessed 2/24/05.
- [11] G. Fedak, C. Germain, V. Neri and F. Cappello, XtremWeb: A Generic Global Computing System, *Proceedings of the 1st IEEE International Symposium on Cluster Computing and the Grid - CCGrid 2001*, May 15-18, 2001, pp. 582-587, Brisbane, Australia.
- [12] Grid.org, "GRID.ORG – Help: Frequently Asked Questions", http://www.grid.org/help/faq_wus.htm, Accessed 4/1/05.
- [13] D. Toth and D. Finkel, A Comparison of Techniques for Distributing File-Based Tasks for Public-Resource Computing, *Proceedings of The 17th IASTED International Conference on Parallel and Distributed Computing and Systems - PDCS 2005*, Phoenix, AZ, U.S.A., November 14-16, 2005, 398-403.
- [14] D. Toth and D. Finkel, Characterizing Resource Availability for Volunteer Computing and its Impact on Task Distribution Methods, *Proceedings of the 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems - SEPADS 2007*, Corfu, Greece, February 16-19, 2007.
- [15] BOINC. "Choosing BOINC projects." <http://boinc.berkeley.edu/projects.php> Updated 8/25/06. Accessed 10/26/06.
- [16] "Work Distribution," http://boinc.berkeley.edu/work_distribution.php Updated 11/11/04, Accessed 2/23/05.
- [17] J. Young, "A First Order Approximation to the Optimum Checkpoint Interval", *Communications of the ACM*, vol. 17, pp. 530-531, September 1974.
- [18] "SETI@home Beta – some questions", http://setiathome.berkeley.edu/forum_thread.php?id=37561, Accessed 2/8/07, Updated 2/5/07.
- [19] "Result Deadline – Unofficial BOINC Wiki", <http://boinc-wiki.ath.cx/index.php?title=Deadline>, Accessed 2/9/07. Updated 2/2/06/
- [20] "Posts by Keck_Komputers", http://einstein.phys.uwm.edu/forum_user_posts.php?userid=2914, Accessed 2/9/07.
- [21] "Folding@Home News", <http://folding.stanford.edu/news.html>, Accessed 2/9/07. Updated 1/22/07.
- [22] "Folding@ Home configuration FAQ", <http://folding.stanford.edu/FAQ-settings.html>, Accessed 2/9/07. Updated 1/1/07.
- [23] "WorkUnits – FaHWiki", <http://fahwiki.net/index.php/WorkUnits>, Accessed 2/8/07. Updated 1/24/07.
- [24] "Workunit size vs. processor", http://einstein.phys.uwm.edu/forum_thread.php?id=4583, Accessed 2/8/07. Updated 7/25/06.
- [25] "Einstein@Home FAQ", <http://einstein.phys.uwm.edu/faq.php>, Accessed 2/8/07.
- [26] "QMC@Home - Wikipedia, the free encyclopedia", <http://en.wikipedia.org/wiki/QMC%40Home>, Accessed 2/8/07.
- [27] "Report deadline too short", http://lhathome.cern.ch/forum_thread.php?id=1977, Accessed 2/8/07. Updated 1/23/06.
- [28] "get wu's with deadline less than 5 days, why??", http://lhathome.cern.ch/forum_thread.php?id=1619. Accessed 2/9/07. Updated 8/30/05.
- [29] "grid.org Forums – View topic – READ ME ==- Work Units (WU)", <http://forum.grid.org/phpBB/viewtopic.php?t=8847&highlight=workunit+size>, Accessed 2/8/07. Updated 1/14/06.

[30] “Rosetta@Home FAQ (work in progress),
http://boinc.bakerlab.org/rosetta/forum_thread.php?id=669, Accessed 10/23/06. Updated 6/10/06.

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

[32] “BOINCSIMAP :: View topic – Wus”,
<http://boinc.bio.wzw.tum.de/boincsimap/forum/viewtopic.php?t=5>, Accessed 2/8/07. Updated 11/29/05.

[33] “The Riesel Sieve Project :: View Topic – Length of WU”,
<http://www.rieselsieve.com/forum/viewtopic.php?t=819>, Accessed 2/9/07, Updated 8/20/06.

[34] “PerlBOINC :: RieselSieve”,
<http://boinc.rieselsieve.com/?faq>, Accessed 2/8/07.

[35] “The Riesel Sieve Project :: View topic – Checkpointing?”,
<http://www.rieselsieve.com/forum/viewtopic.php?t=1084>, Accessed 2/9/07, Updated 1/19/07.

[36] “World Community Grid – View Thread – Run times for work units – what to expect”,
<http://worldcommunitygrid.org/forums/wcg/viewthread?thead=928>, Accessed 2/8/07. Updated 12/10/04.