Celebrating Diversity in Volunteer Computing

David P. Anderson
University of California, Berkeley
davea@ssl.berkeley.edu

Kevin Reed
IBM
knreed@us.ibm.com

Abstract

The computing resources in a volunteer computing system are highly diverse in terms of software and hardware type, speed, availability, reliability, network connectivity, and other properties. Similarly, the jobs to be performed may vary widely in terms of their hardware and completion time requirements. To maximize system performance, the system's job selection policy must accommodate both types of diversity.

In this paper we discuss diversity in the context of World Community Grid (a large volunteer computing project sponsored by IBM) and BOINC, the middleware system on which it is based. We then discuss the techniques used in the BOINC scheduler to efficiently match diverse jobs to diverse hosts.

1. Introduction

Volunteer computing is a form of distributed computing in which the general public volunteers processing and storage resources to computing projects. BOINC is a software platform for volunteer computing [2]. BOINC is being used by projects in physics, molecular biology, medicine, chemistry, astronomy, climate dynamics, mathematics, and the study of games. There are currently 50 projects and 580,000 volunteer computers supplying an average of 1.2 PetaFLOPS.

Compared to other types of high-performance computing, volunteer computing has a high degree of diversity. The volunteered computers vary widely in terms of software and hardware type, speed, availability, reliability, and network connectivity. Similarly, the applications and jobs vary widely in terms of their resource requirements and completion time constraints.

These sources of diversity place many demands on BOINC. Foremost among these is the **job selection problem**: when a client contacts a BOINC scheduling server, the server must choose, from a database of perhaps a million jobs, those which are "best" for that client according to a complex set of criteria. Furthermore, the server must handle hundreds of such requests per second.

In this paper we discuss this problem in the context of the IBM-sponsored World Community Grid, a large BOINC-based volunteer computing project. Section 2 describes the BOINC architecture. Section 3 summarizes the population of computers participating in World Community Grid, and the applications it supports. In Section 4 we discuss the techniques used in the BOINC scheduling server to efficiently match diverse jobs to diverse hosts.

This work was supported by National Science Foundation award OCI-0721124.

2. The BOINC model and architecture

The BOINC model involves **projects** and **volunteers**. Projects are organizations (typically academic research groups) that need computing power. Projects are independent; each operates its own BOINC server.

Volunteers participate by running BOINC client software on their computers (**hosts**). Volunteers can **attach** each host to any set of projects, and can specify the quota of bottleneck resources allocated to each project.

A BOINC server is centered around a relational database, whose tables correspond to the abstractions of BOINC's computing model:

- **Platform**: an execution environment, typically the combination of an operating system and processor type (Windows/x86), or a virtual environment (VMWare/x86 or Java).
- **Application**: the abstraction of a program, independent of platforms or versions.
- **Application version**: an executable program. Each is associated with an application, a platform, a version number, and one or more files (main program, libraries, and data).
- **Job**: a computation to be done. Each is associated with an application (not an application version or platform) and a set of input files.
- **Job instance**: the execution of a job on a particular host. Each job instance is associated with an application version (chosen when the job is issued) and a set of output files.

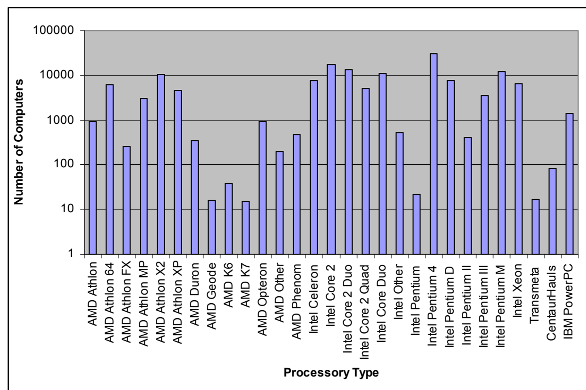
The results returned by volunteers are not always correct, due to hardware malfunction (particularly on over-clocked computers) or to malicious volunteers attempting to undermine the project or to get credit for computing not actually performed. As a result, **result validation** is typically required.

BOINC supports several techniques for result validation; the most basic is **replicated computing**, in which instances of each job are sent to two different hosts. If the results agree, they are assumed to be correct. Otherwise further instances are issued until a consensus of agreeing results is obtained or a limit on the number of instances is reached.

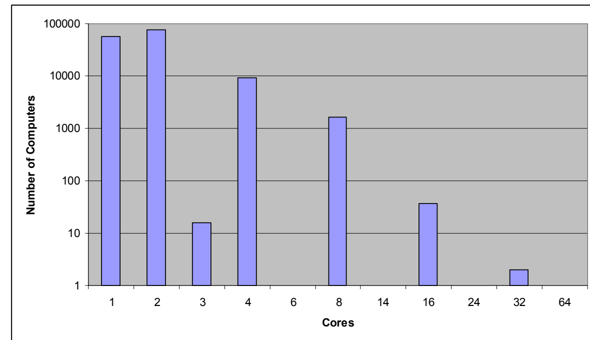
3. Host and application diversity in World Community Grid

World Community Grid is a large-scale volunteer computing project sponsored by IBM with over 390,000 volunteers, who have contributed over 167,000 years of processing time since its launch in November 2004. It is currently producing an average power of 170 TeraFLOPS from volunteered computers.

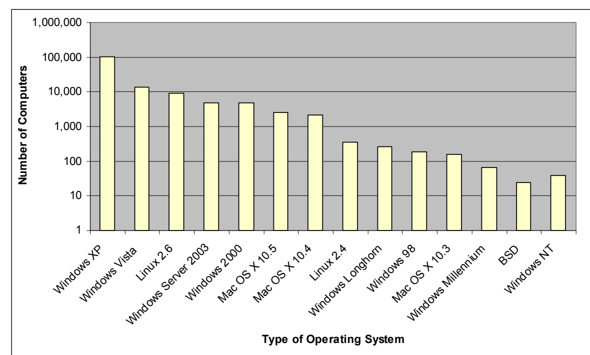
The computers attached to World Community Grid vary widely in terms of processor, operating system, memory, disk, network bandwidth, availability, error rate, and turnaround time. Figure 1 shows the distributions of these properties.



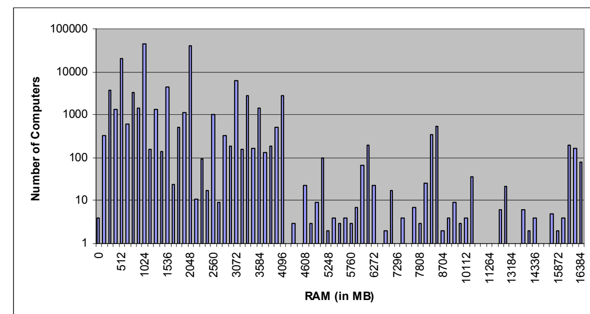
a) Processor type



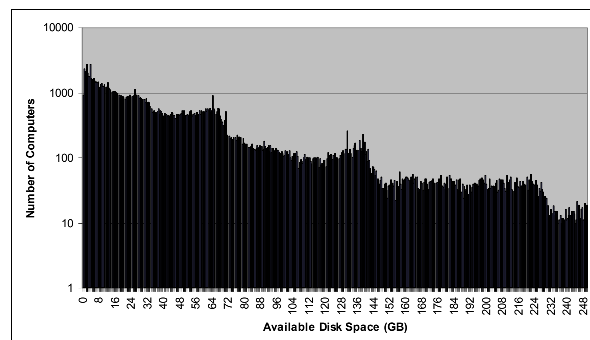
b) Number of cores



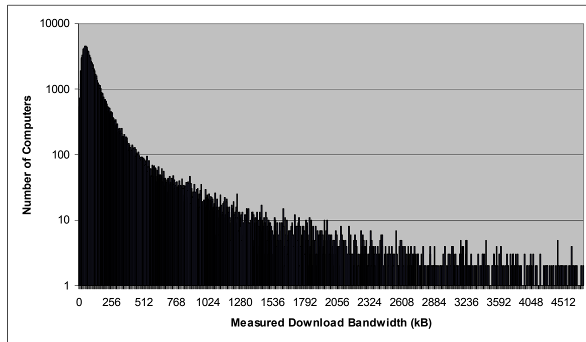
c) Operating system



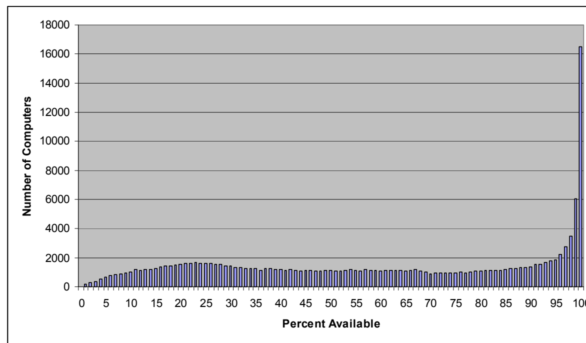
d) Physical memory



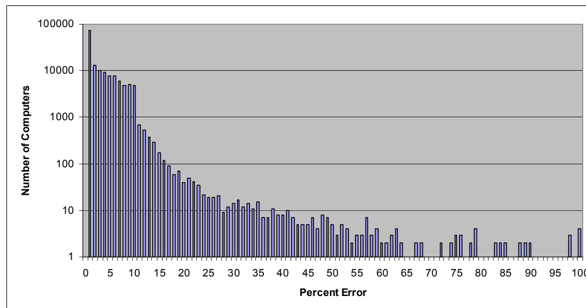
e) Available disk space



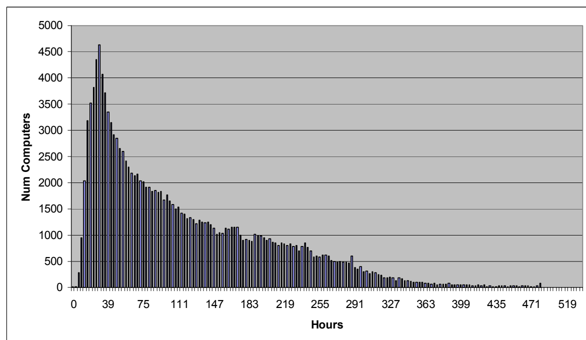
f) Download bandwidth



g) Availability



h) Job error rate



i) Average turnaround time

Figure 1: Properties of hosts attached to World Community Grid

At the time of this writing, World Community Grid has six active applications, which have varying requirements for disk space, computational power, memory and bandwidth, as shown in Figure 2.

	Min Download Bandwidth (kB/s)	Input File Size, First Job (MB)	Input File Size, Subsequent Jobs (MB)	Result File Size (MB)	Disk Space (MB)	Memory (MB)
AfricanClimate@Home	65	180	180	121	1,000	500
Discovering Dengue Drugs - Together	0	1	0.2	0.7	250	125
FightAIDS@Home	0	0.2	0.2	0.2	200	125
Help Conquer Cancer	0	0.1	0.1	0.1	50	100
Human Proteome Folding - Phase II	0	8	2	0.3	200	180
Nutritious Rice for the World	0	5	0.3	1	32	50

Figure 2: Resource requirements of World Community Grid applications

Different result validation techniques are used, in accordance with the properties of the applications.

The Human Proteome Folding and Nutritious Rice applications do not use replication. The statistical distribution of results is known in advance, and invalid results appear as outliers. To detect submission of duplicate results, each job includes a small unique computation that can be checked on the server.

The Help Conquer Cancer, FightAIDS@Home and Discovering Dengue Drugs applications use replicated computing.

4. Job selection in BOINC

When a BOINC client is attached to a project, it periodically issues a **scheduler request** to the project's server. The request message includes a description of the host and its current workload (jobs queued and in progress), descriptions of newly-completed jobs, and a request for new jobs. The reply message may contain a set of new jobs. Multiple jobs may be returned; this reduces the rate of scheduler requests and accommodates clients that are disconnected from the Internet for long periods.

A BOINC project's database may contain millions of jobs, and its server may handle tens or hundreds of scheduler requests per second. For a given request we'd ideally scan the entire job table and send the jobs that are "best" for the client according to criteria to be discussed later. In practice this is infeasible; the database overhead would be prohibitive.\

Instead, BOINC uses the following architecture (see Figure 3):

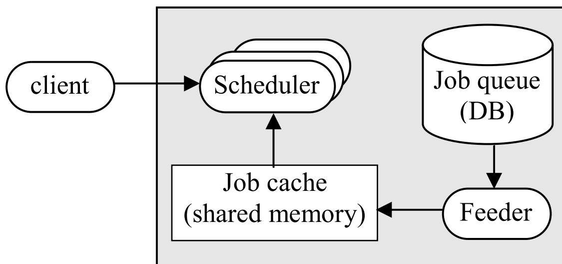


Figure 3: BOINC server architecture

- A cache of roughly 1000 jobs is maintained in a shared memory segment.
- This cache is periodically replenished by a "feeder" program that enumerates jobs from the database.
- The scheduler runs as an Apache CGI program; at a given moment there may be dozens or hundreds of instances of it running. Each instance scans the jobs in cache, and returns the best ones.

This design provides high performance; a server can issue hundreds of jobs per second [1]. It also provides the needed support for diversity: if the job cache is sufficiently large, it is probable that it will contain jobs well-suited to a given client.

The "baseline" job selection policy is:

- Scan the job cache, starting at a random point. For each job, do feasibility checks that do not require database access: for example, check that the host has sufficient RAM and disk space, and that it is likely to complete the job by its deadline.
- If the job passes these checks, lock it, and do checks that require database access: for example, that no other instance of the same job has already been issued to a host belonging to the same volunteer.
- Continue until sufficient jobs have been selected to satisfy the host's work request.

In the remainder of this section we describe refinements to this baseline policy that deal with various types of diversity.

4.1. Multiple platform mechanism

In early versions of BOINC, a client had a single platform. It reported this platform in scheduler request messages, and the scheduler sent it application versions for that platform.

This model broke down with the release of Intel-based Macintosh computers, which were able to execute PowerPC binaries via emulation. Should such a host advertise itself as Intel/Mac? If so it would be given no work by projects that had only PowerPC/Mac executables. Similarly, Linux/x64 and Windows/x64 hosts are generally able to execute Linux/x86 and Windows/x86 binaries as well; and some BSD-based systems can execute Linux binaries.

To deal with these situations we allow a host to support multiple platforms; the BOINC client generates this list by examining the host at runtime. Scheduler request messages contain a list of platforms in order of decreasing expected execution speed, and the scheduler sends applications for the fastest expected platform.

4.2. Co-processors and multi-core

Originally BOINC assumed that an application used a single CPU. On a multiprocessor, the client would run one application per CPU. More recently, BOINC has been extended to handle applications that use multiple CPUs and/or use co-processors such as graphics processing units (GPUs) and the SPEs in a Cell processor.

This introduces a new challenge for job selection: if several variants of an application are available for a given platform, which one should be sent to a particular host? This decision may require application-specific knowledge, such as the speedup of a parallel application as a function of the number of CPUs.

BOINC addresses this as follows. A project may have several application versions for a given platform and platform (for example, a single-threaded version, a multi-threaded version, and a GPU-enabled version). Each version is tagged with a "planning class".

The BOINC client has been extended to detect the presence of particular co-processor types (e.g. CUDA-enabled GPUs [8]) and to report them in scheduler request messages.

The scheduler is linked with a project-supplied **application planning function**. This function is passed a planning class and a host description, including CPU and co-processor information. It returns the expected FLOPS that an application of the given class will achieve on the host, the average number of CPUs that it will use, and the set of co-processors that it will use. For a given job, the scheduler examines the available application versions, and picks the one for which the expected FLOPS is highest.

The scheduler reply includes the resource usage for each application version; the client takes this into account when scheduling applications (for example, it won't try to run two applications that both need a GPU if only one is available).

4.3. Volunteer application selection

A project may have multiple applications. By default jobs are assigned without regard to their application. However, several mechanisms are available to limit or control job assignment on the basis of application.

First, projects can specify that jobs are to be dispatched with given ratios between applications, e.g. that of every 10 jobs dispatched, 6 should be of application 1 and 4 should be application 2. This is enforced by the feeder, which uses separate database enumerations for the different applications, and interleaves jobs in the cache according to the ratios.

Second, projects can allow volunteers to select particular applications, and they will be sent only jobs for those applications (volunteers may also select an option saying that they will accept jobs for any application in the case where no jobs for their selected applications are available).

Finally, applications may be designated as "beta test", and volunteers can select a "beta tester" option. A beta tester will be sent jobs from beta-test applications if any are available.

4.4. Handling numerical variability

The validation mechanism described in Section 2 require the ability to decide when two results are equivalent. This can be difficult because of numerical variation between hosts; differences in processor types, compiler output, and math libraries can all contribute to numerical discrepancies.

If an applications is numerically stable it is usually possible to define a "fuzzy comparison" function. However, for unstable applications (such as simulations of chaotic systems) this is not possible since small deviations lead to unboundedly large differences in results.

To address this problem, BOINC provides a mechanism called **homogeneous redundancy (HR)** [6]. A project can define one or more **HR types**. Each HR type consists of an equivalence relation on the set of hosts, based on their OS and CPU, as computed by a customizable function (BOINC supplies some default HR types, but projects can define their own).

An application can be associated with an HR type; for hosts that are equivalent in the HR type, the application is assumed to compute bit-identical results.

Once an instance of a job has been sent to a host, the scheduler will send further replicas of that job only to hosts that are equivalent in the application's HR type. In this case, the job is said to be **committed** to that HR class.

This introduces a problem: the job cache may fill up with jobs that have been committed to a particular HR class. When hosts of other HR classes request work, none will be available. We address this problem as follows:

- A "Census" program periodically enumerates all hosts in the database, and builds a table that, for each HR class, gives the recent average credit granted to hosts in that class.
- For each HR class, a certain number N of slots in the job cache are reserved for jobs that are committed to that class; N is proportional to the classes' recent average credit, and is at least 1.
- A fraction of the slots in the job cache are reserved for jobs that are not committed.

As the feeder enumerates jobs J from the database, it must check whether the allocation for J's HR class would be exceeded; if so it doesn't add J to the cache.

4.5. Job size matching

The throughput of a host is the product of its processor speed and its availability; as shown in Section 3, this varies by two orders of magnitude in World Community Grid.

For many applications, the job size (i.e. the number of FLOPS per job) can be set arbitrarily. How should a project set its job size? If job size is too large, slow hosts won't be able to complete jobs by their deadline; if it's too small, load on the server will be excessive.

Ideally we'd like to be able to pick a certain interval T between scheduler requests (say, 1 day) and then send each host a job that will take time T to complete. Achieving this goal requires that:

- The project must generate an appropriate distribution of job sizes;
- The BOINC scheduler must take job size into account.

The second goal is accomplished by having the Census program (Section 4.4) compute the mean and standard deviation of host throughputs. The Feeder reads this and stores it in the shared memory segment. The Feeder also periodically computes the average and

standard deviation of the job sizes currently in shared memory.

Given a host whose throughput differs by X standard deviations from the mean, the scheduler then attempts to send it jobs that differ by approximately X standard deviations from the job-size mean.

4.6. Adaptive replication

As shown in Section 3.5, hosts have varying error rates, with a majority of hosts having a near-zero error rate. While replicated computing is necessary to establish the trustworthiness of hosts, it is inefficient thereafter.

BOINC offers the option of **adaptive replication**, in which the scheduler maintains a dynamic estimate of the error rate E(H) of each host. If E(H) is above a constant K, then all jobs to that host are replicated. If E(H) < K, then jobs to H are replicated randomly, with a probability that goes to zero as E(H) approaches zero. The initial estimate of E(H) is sufficiently high that a host must complete some number of jobs correctly before it is eligible for non-replication.

This policy does not eliminate the possibility of erroneous results, but with appropriate parameters it can reduce the incidence of errors to a required level while at the same time reducing the overhead of replication.

4.7. Finishing retries quickly

If a job instance times out or cannot be validated, an additional "retry" instance is created. Only when the retry is completed can other instances be granted credit, and the job files deleted from the server. Thus it is desirable to finish retries quickly.

To do this, the BOINC identifies hosts that have a history of returning valid results quickly. Retries are then sent preferentially to these hosts. Since these hosts are also known to return work quickly, the deadline for the job instance is also reduced to ensure that the work is processed and returned quickly.

4.8. Score-based scheduling

In the preceding sections we have listed a number of criteria for job assignment, corresponding to different kinds of host and job diversity. How can these criteria be combined into a single job assignment policy?

This is done using a **scoring function** *S(J, H)*. The job assignment policy is essentially to send H the cached jobs J for which S(J, H) is greatest. The default scoring function is a linear combination of the following terms:

- The expected FLOPS of the fastest application version available for H (try to send jobs for which fast applications are available)
- 1 if the volunteer selected J's application, else 0 (try to give volunteers jobs for the applications they selected).
- 1 if J is already committed to the homogeneous redundancy class to which H belongs, else 0 (try to finish committed jobs rather than create new ones).
- $-(A-B)^2$, where A is the size of the job and B is the speed of the host, expressed in units of standard deviation (see Section 4.5).
- The disk and RAM requirements of J (avoid sending jobs that need few resources to hosts that have lots of resources).
- 1 if we've decided to send H a non-replicated job,and J has no other replicas, else 0.

Projects can adjust the weights of these terms, or they can replace the scoring function entirely.

4.9. Assigned jobs

Some projects may need to make scheduling decisions based on information that BOINC doesn't know about – for example, whether the client has a particular software package installed.

To handle cases like this, BOINC allows projects to provide "probe jobs", which are executed exactly once on all hosts. The output of this job (typically an XML document) is stored in the host record in the database, and can be referenced in a project-specific scoring function (Section 4.8).

4.10. Host punishment

Some hosts, for various reasons, experience errors immediately for all job instances that they attempt to process. These hosts can then get into a cycle where they request job instances from the server, experience an error and then notify the server of their error within a few minutes. This can lead to a host downloading a significant amount of data in a short period as it repeatedly fetches new work and reports errors.

BOINC addresses this problem by allowing projects to set a limit on the number of job instances that can be downloaded by a specific host during one day (the limit scales with the number of processors

available on the host). In the event that a host reaches its limit for the day, then it cannot download more job instances until the following day. Additionally, if a host reports a job instance that experienced an error then that host's daily limit is reduced by one. Thus a host who experiences nothing but errors will be eventually limited to only one job instance per day. Once the problem has been corrected on the host, it will start to return successful job instances. As successful job instances are returned, then the limit is doubled until it reaches the limit set by the project.

4.11. Anonymous platform mechanism

Up to this point, we've assumed that projects supply executables to volunteers. What if a project makes the source code for its applications available (e.g., they are open source)? Then volunteers can compile the applications themselves. There are several reasons why volunteers might want to do this:

- Their computer's platform is not supported by the project.
- For security reasons, they are willing to run only programs that they have security-audited and compiled from source;
- To increase performance, they want to compile the application with a particular compiler or set of compiler options;
- To increase performance, they want to modify the application (they may do this incorrectly, but that will be detected by BOINC's replication mechanism).

BOINC supports user-compiled applications with its **anonymous platform mechanism**. The basic idea is that the client's scheduler requests, instead of specifying a platform, provide a list of application versions present on the host. The server sends jobs that can use these application versions (but doesn't send new application versions).

To use this mechanism, the volunteer must create an XML file describing the available application versions, and put it at a particular place in the BOINC directory hierarchy.

5. Related work

A number of frameworks for volunteer computing have been developed; examples include Bayanihan [7], Xtremweb [3] and Entropia [4]. Each of these frameworks addresses some of the issues we have discussed; for example, Bayanihan uses a result validation technique called **spot-checking**, which is similar to BOINC's adaptive replication. However, none of these systems has been been deployed on the scale of World Community Grid or other large BOINC projects, nor has any of them been used for as large a range of applications.

Folding@home [5] is a large (~200,000 node) volunteer computing project, which has developed its own infrastructure. It supports diverse resources, including GPUs and Sony Playstation 3 game consoles. Details of its infrastructure have not been published, but it uses a two-level scheduler in which clients contact a "master scheduler", which directs them (perhaps based on their properties) to one of a number of job queues, corresponding to different applications or job types.

Another interesting comparison is with Condor's ClassAds mechanism [9]. In this system, job queue nodes supply "help wanted" ads and worker nodes supply "help offered" ads;. The ads are written in an extensible notation that includes a scripting capability, and can include information about host resources and load conditions, organizational identity, and so on. A "matchmaker" server matches compatible ads and directs the worker node to contact the job queue node.

The differences between ClassAds and the mechanisms described here stem from differing basic assumptions and goals between BOINC and Condor:

- Condor assumes that worker nodes are trusted and have negligible error rate. There is no need for result validation, and there is no need for servers to maintain error rates or other information about worker nodes.
- BOINC is designed to handle scale two or three orders of magnitude larger than Condor.
- Condor assumes that worker nodes are constantly connected, and has a mechanism for migrating jobs. Therefore workers don't queue jobs, and scheduling policies don't involve work in progress. In BOINC, on the other hand, client can queue jobs – both because of multiple project attachments and because of sporadic Internet connection – and the scheduling policy must reflect current workload.

6. Conclusion

We have described the problem of matching diverse jobs and diverse hosts, and have described how BOINC addresses this problem.

These mechanism are operational on World Community Grid and other BOINC projects, and they seem to work. However, we currently are unable to quantify how well they work. It is difficult to do controlled experiments in the context of a large volunteer computing project, because there are many factors that cannot be controlled, and because poorly-performing mechanism can waste lots of resources and perhaps drive away volunteers.

To remedy this situation, we are currently developing a simulator that will allow us to study server policies. The simulator uses trace-driven or statistical models of host and volunteer populations, including factors such as host churn, availability, and reliability. The server part uses the actual BOINC production code, adapted to use simulated time instead of real time, for maximal fidelity.

Several of the mechanisms described here are at an early stage of development. For example, the support for multi-CPU and co-processor applications sends the application version that will run fastest in isolation. This is not necessarily optimal – it may be better to send one CPU job and one GPU job (since they can execute concurrently) rather than two GPU jobs.

7. References

[1] Anderson, D.P., E. Korpela, and R. Walton. High-Performance Task Distribution for Volunteer Computing. First IEEE International Conference on e-Science and Grid Technologies. 5-8 December 2005, Melbourne

[2] Anderson, D.P. “BOINC: A System for Public-Resource Computing and Storage”. 5th IEEE/ACM International Workshop on Grid Computing, pp. 365-372, Nov. 8 2004, Pittsburgh, PA.

[3] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Neri and O. Lodygensky. Computing on Large Scale Distributed Systems: XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid. FGCS Future Generation Computer Science, 2004.

[4] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. J. Parallel Distrib. Comput. 63(2003) 597-610.

[5] S.M. Larson, C.D. Snow, M. Shirts and V.S. Pande. “Folding@Home and Genome@Home: Using distributed computing to tackle previously intractible problems in computational biology”. Computational Genomics, Horizon Press, 2002.

[6] Taufer, M., D. Anderson, P. Cicotti, C.L. Brooks III. “Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results Using Public Computing”. Heterogeneous Computing Workshop, International Parallel and Distributed Processing Symposium 2005, Denver, CO, April 4-8, 2005.

[7] Sarmenta, L.F.G. and S. Hirano. “Bayanihan: Building and Studying Web-Based Volunteer Computing Systems Using Java”. Future Generation Computer Systems, 15(5/6), 1999.

[8] Schatz, M.C., Trapnell, C., Delcher, A.L., Varshney, A. (2007). High-throughput sequence alignment using Graphics Processing Units. BMC Bioinformatics 8:474.

[9] Raman, R, M. Livny, and M. Solomon, Matchmaking: Distributed Resource Management for High Throughput Computing, IEEE HPDC'98, pp. 140-147, July 1998.