Massey University
College of Sciences

BOINC

COMMUNITY GRID COMPUTING

May 2009

By: Dragan Zakic
06235808

Course Coordinator: Dr. Martin Johnson

Paper: Studies in Parallel and Distributed Systems
159.735

ABSTRACT

Projects which require extreme computing power no longer look at supercomputing centres, but rather at hundreds of millions of personal computers interconnected and distributed all over the world. This report looks at various aspects of volunteer community grid computing, the potential it has and challenges it is faced with. The objective of the report is to present the current state of this discipline as well as the current leader in the field. In addition, it highlights typical problems that are suitable for deployment through community grids like BOINC and ways to break the barrier to entry.

TABLE OF CONTENTS

1. INTRODUCTION

In general, greater computing power allows simulations with closer approximation of reality. This has driven the development of computers that are as fast as possible.

Moore's Law asserts that the speed of CPU chips doubles about every 18 months. The rate of progress is even faster for graphics coprocessors. Their doubling time is about 8 months, and current graphics chips have a raw floating point arithmetic speed many times that of their host CPU.

In the 1990s two important things happened. First, because of Moore's Law, PCs became very fast - as fast as supercomputers only a few years older. Second, the Internet expanded to the consumer market. Suddenly there were millions of fast computers, connected by a network. The idea of using these computers as a parallel supercomputer occurred to many people independently. Two projects of this type emerged in 1997: GIMPS, which searched for large prime numbers, and Distributed.net, which deciphers encrypted messages.

These projects attracted thousands of participants. In 1999, a third project, SETI@home, was launched, with the goal of detecting radio signals emitted by intelligent civilizations outside Earth. SETI@home acts as a "screensaver", running only when the PC is idle, and providing a graphical view of the work being done. The project's appeal extended beyond hobbyists; it attracted millions of participants from all around the world. It inspired a number of other academic projects, as well as several companies that sought to commercialize the public computing paradigm.

1.1. TECHNICAL ASPECTS OF PUBLIC COMPUTING

Conducting a public computing project requires adapting an application program to various platforms, implementing server systems and databases, keeping track of user accounts and credit, dealing with redundancy and error conditions, as well as numerous other tasks.

Berkeley Open Infrastructure for Network Computing is developed with the goal to make it easy and cheap to convert an existing application to a public computing project. BOINC solves or helps solve most of these problems.

BOINC started as a project at Berkeley, University of California. Dr. David Anderson and his team created a system that allows volunteers from around the world to safely donate their spare computing resources to scientific research. BOINC projects are autonomous; each one maintains its own servers and databases, and does not depend on others. Participants can register with multiple projects, and can control how their resources are shared.

1.2. THE POWER OF PUBLIC COMPUTING

Public computing can provide more computing power than any supercomputer, cluster, or grid, and the disparity will grow over time.

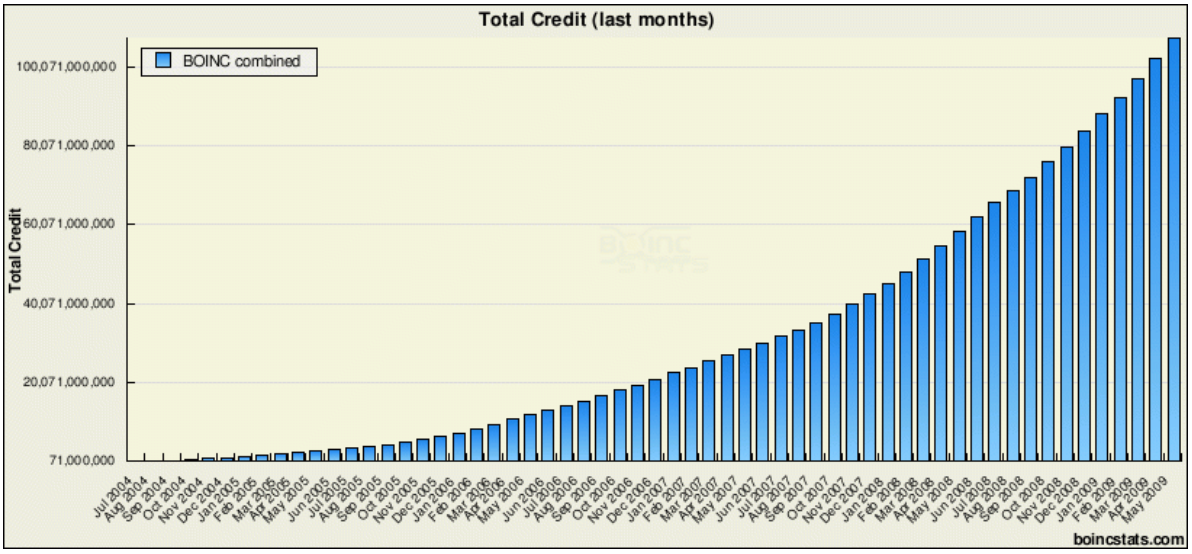


TABLE 1: BOINC COMBINED COMPUTING POWER

As of May 2009, the total computing power of entire BOINC grid averages over 1.7 PFLOPS on about 550,000 active computers, and keeps growing exponentially.

Folding@home, a Stanford University project that studies the folding of proteins, is of May 2009 sustaining over 8 PFLOPS. It is the first computing project of any kind to cross the four petaflops milestone. This level of performance is primarily enabled by the cumulative effort of a vast array of PlayStation 3 and powerful GPU units.

OS Type	Native TFLOPS*	x86 TFLOPS*	Active CPUs	Total CPUs
Windows	275	275	289268	2682514
Mac OS X/PowerPC	5	5	5678	127521
Mac OS X/Intel	23	23	7316	88212
Linux	53	53	30992	387321
ATI GPU	967	1020	9480	61731
NVIDIA GPU	2284	4819	19193	110557
PLAYSTATION®3	1039	2192	36856	795259
Total	4646	8387	398783	4253115

TABLE 2: FOLDING@HOME STATS

Folding@home is not a BOINC project, though it utilises the same principles of distributed community computing as BOINC does. The founders have actively considered porting the

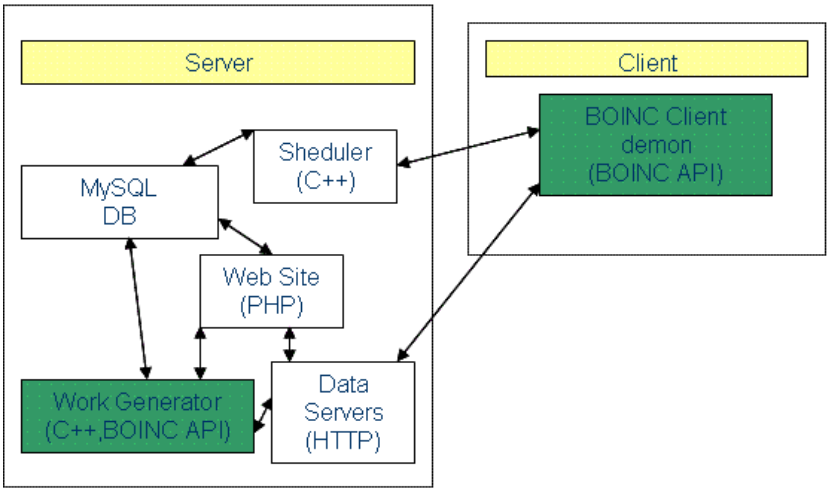
project to BOINC platform; however being a mature project with huge existing member base has made the transition difficult.

By comparison, the latest upgrade to Cray XT Jaguar supercomputer at the U.S. Department of Energy in November 2008 has increased the system's computing power to a peak 1.64 petaflops. The first to pass one petaflops was IBM's Roadrunner in May 2008, running 6,480 AMD Opterons in tandem with 12,960 IBM PowerXCell 8i processors.

These expensive supercomputing powers are funded, maintained and used by government agencies. Reserving these kinds of resources for scientific research at universities may prove costly; so community computing comes as great cost effective alternative.

2. BOINC ARCHITECTURE

BOINC is designed to be a free structure for anyone wishing to start a volunteer computing project. It is a set of software modules that enable the use of idle CPU and GPU cycles on a personal computer to do scientific computing.



The server generates work units, distributes them to clients and collects their results. Each PC, acting as a client, communicates with the server to get work units which include executables and input files and return results of computation. The clients do not communicate among themselves.

BOINC implementation successfully solves the difficulties of unstable network, unstable clients who may join or leave the grid at any time.

2.1. BOINC SERVER

A major part of BOINC is the backend server which runs on Linux and use Apache, PHP, and MySQL as a basis for its web and database systems, which easily scales to projects of any size. BOINC servers also provide these features:

1. Homogenous redundancy (sending work units only to computers on the same platform –operating system, CPU type, etc.)
2. Work unit trickling (sending information back to the server before the work unit completes)
3. Locality scheduling (can send work units to computers that already have the necessary files and creates work on demand)
4. Work distribution based on host parameters (work units requiring certain amount of RAM, etc)

The server consists of two CGI programs and usually five daemons, written in C++. Computations to be performed by clients are called wok units. A result describes an instance of a work unit, even if it hasn't been completed. A project does not explicitly create results; the server creates them automatically from work units.

All BOINC components come ready available to use. The project owner need only provide the work generator process.

SCHEDULER

The scheduler program handles requests from clients, receiving completed results and sending new work to compute. The reader periodically fills empty slots in the shared memory region after the scheduler has sent those work units to a client.

FEEDER

The scheduler does not get available work units directly from the database. Instead, there is a feeder daemon that loads tasks from the database, and keeps them in a shared memory block, which the scheduler reads.

TRANSITIONER

The transitioner is responsible of managing the state transitions of work units. The transitioner decides when to send the work units, checks the results for errors, and determines when the work units can be deleted.

VALIDATOR

One of the main reasons BOINC managed to sustain large number of public participants is the credit reward scheme, where members are unaccountable, but still recognised for their contribution. To prevent cheating, BOINC server will periodically assign the same work unit to more than one client and validate the result once both have completed.

The validator can have custom project code to do fuzzy comparison between results, or it can be just a bitwise comparison. If the results match, the work unit is marked as valid, users are granted credit for it, and a "canonical result" is chosen.

ASSIMILATOR

Next, the assimilator daemon processes the canonical result using project-specific code. An assimilator may also generate more work units based on the returned data.

CRITICISM

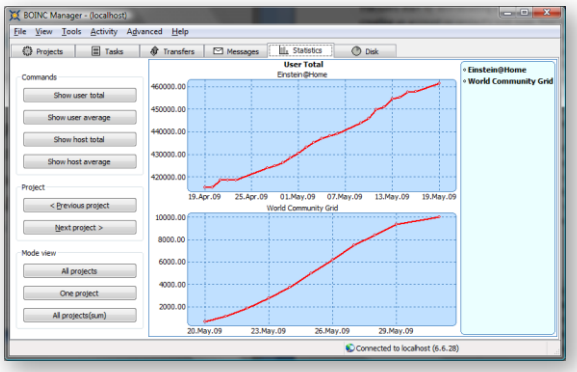
The BOINC server implementation has been widely criticises for being complicated to setup, hard to maintain, and having high load on the servers that host the project in terms of bandwidth and utilisation.

Furthermore, it is designed to be deployed on Unix-like systems only. It can run on XP and Vista systems, but the design structure makes this difficult and more expensive.

Servers are not really as simple to deploy as the client as they are based on a large number of scripts. The server project website does a poor job storing a compiled database of server side scripts for those wishing to create a BOINC project

2.2. BOINC CLIENT

The users start by downloading a 6.5MB BOINC client, the one used by all BOINC projects. After creating an account on project's web page, they chose a "Join Project" option and the client begins downloading the initial task code and data, does some initial benchmarking, and finally the number crunching begins. The user can join as many projects as he/she wishes, and decides on the resource ratio allocated to each project. The BOINC client is carefully written in such manner that it runs in the lowest available priority mode. This allows the user to continue using the computer with all the available computing power still available to user applications on demand. In practice, the CPU usage will show flat 100% usage; indicating there is no CPU cycle wasted. The memory consumption however will be higher than usual which can cause some extra paging and inherently slow down the performance.



BOINC client empowers users with plenty of configuration options to choose from, including the network bandwidth the client is allowed to use, number of CPUs, time schedule, and percentage of CPU time available to BOINC tasks. To further reward users for their perceived disadvantage, the project team is encouraged to publish daily statistics with top contributing users, countries,

and teams. But mainly, participants donate their computing power for they are genuinely interested in scientific research and feel good about being able to take part and make contribution to humanity.

API

BOINC client therefore provides an environment for the number crunching code to run in isolation, without having to worry whether the time is right, and whether there is enough work queued up. The project client code is written in C or C++, using BOINC API library which abstracts away all above issues. The code can safely focus on solving the main problem and providing the answer to the work unit.

All applications must initialise BOINC environment with a call to:

```
int boinc_init();
```

When completed, they must call

```
boinc_finish(int status);
```

To convert logical file names to physical names, for example, instead of

```
f = fopen("my_file", "r");
```

The application might use

```
string resolved_name;
retval = boinc_resolve_filename_s("my_file", resolved_name);
if (retval) fail("can't resolve filename");
f = fopen(resolved_name.c_str(), "r");
```

The core client GUI displays the percent done of work units in progress. To keep this display current, an application should periodically call:

```
boinc_fraction_done(double fraction_done);
```

Call these around code segments during which you don't want to be suspended or killed by the core client.

```
void boinc_begin_critical_section();
void boinc_end_critical_section();
```

Computations that use a significant amount of time per work unit may want to periodically write the current state of the computation to disk whenever an application reaches a point where it is able to checkpoint.

```
int boinc_time_to_checkpoint();
```

If this returns nonzero (True) then the application should checkpoint immediately (i.e., write the state file and flush all output files), then call

```
void boinc_checkpoint_completed();
```

3. APPLICATIONS OF PUBLIC COMPUTING

A task suitable to public computing must be divisible into independent pieces whose ratio of computation to data is high. Otherwise, the cost of Internet data transfer may exceed the cost of computing centrally. Many types of computations have these properties:

1. Complex physical systems with a random and chaotic component. Their outcome is probabilistic, not exact. Studying the statistics of this outcome requires running large numbers of simulations with different random initial and boundary conditions. These simulations can be run in parallel.
2. There is an evolving field of "random algorithms" that provide approximate solutions to exact problems. These often involve random trials that can run in parallel.
3. Genetic algorithms are applicable to many areas. This approach involves creating a population of approximate solutions to a problem, and using the mechanisms of natural selection to approach an optimal solution.
4. Models of physical systems often have large numbers of underlying parameters whose optimal values are not known, and which combine nonlinearly. Exploring such parameter spaces requires large numbers of independent simulation runs. More generally, "Monte Carlo" algorithms involve large numbers of independent computations, corresponding to sampling in a high-dimensional space.
5. Applications that involve analyzing large amounts of data, such as data from a radio telescope (e.g., SETI@home) or from a particle accelerator, have inherent parallelism. The limiting factor is the computation-to-data ratio.
6. Some medical projects involve searching a set of millions or billions of molecules (for example, searching for potential drugs). These tasks are easily parallelized. Similarly some genetics projects involve matching a set of proteins with a DNA sequence; again, this is easily parallelized.

4. CONCLUSION

Public computing may help bring the public closer to science and scientific research. If computer owners can donate their resources to any of a wide range of projects, they will study and evaluate these projects, learning about their goals, methods, and chances of success. This process might be further encouraged by the creation of "decision markets" in which the public can make virtual bets or investments based on the outcome of science projects, analogously to political decision markets. Because computer owners can contribute to whatever project they choose, the control over resource allocation for science will be shifted away from government funding agencies and towards the public. This has its risks: the public may be easier to deceive than a peer-review panel. But it offers a very direct and democratic mechanism for deciding research policy.

Many scientific ideas for computations that were initially thought to take a million years of computer time may now be taken out of the wastebasket and reconsidered.

The overhead of setting up and maintaining a BOIC server remains a high barrier to entry for many academic institutions. To keep the projects going, substantial amount of volunteer time is required to keep the forums alive, maintain statistics pages, and keep server modules running. Still, it is the best platform available which solves myriad of problems involved with community computing out of the box.

One way to get quickly up and going is to use publicly available, preconfigured, virtual machine image with server software preinstalled. It is a 900MB Debian-based Linux distribution, last updated February 2009 and contains all necessary software loaded and configured, ready to compile the project and start serving clients.

What is not included in the package is a good project idea that has great public appeal, Internet access to the server, and BOINC client and server code.

Yet, all difficulties aside, BOINC platform offers unprecedented low cost per node large scale computing for many academic institutions, which has unlimited potential to grow.

5. REFERENCES

1. Public Computing: Reconnecting People to Science, Dr. David P. Anderson, http://www.it.uom.gr/mpiweb/BOINC/PublicComputingReconnectingPeopletoScience.pdf
2. http://boinc.berkeley.edu
3. http://en.wikipedia.org/wiki/BOINC_client-server_technology
4. http://setiathome.berkeley.edu/
5. http://www.boinc-wiki.info/BOINC_System_Architecture
6. http://www.boinc-wiki.info/BOINC_System
7. http://climateprediction.net
8. http://folding.stanford.edu
9. http://www.globalgridforum.com/
10. http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats
11. http://news.bbc.co.uk/1/hi/technology/7443557.stm
12. http://are.ehibou.com/boinc-boinc-api/
13. http://www.spy-hill.com/~myers/help/boinc/boinc-on-windows.html
14. http://boinc.berkeley.edu/trac/wiki/ServerIntro
15. http://lccb.scripps.edu/tutorials/talk_michela_taufer_090204.pdf
16. http://en.wikipedia.org/wiki/Distributed_computing
17. http://boincstats.com/stats/project_graph.php?pr=bo
18. http://boinc.berkeley.edu/trac/wiki/VmServer
19. http://glue.yahoo.com/page/boinc
20. http://foldingforum.org/