An overview on public resources computers: BOINC

Linda Ponta

Abstract—**BOINC (Berkeley Open Infrastructure for Network Computing) is a middleware system for volunteer computing. This software system makes easy for scientists to create and operate public-resource computing projects. BOINC supports diverse applications, including those with large storage or communication requirements. PC owners can participate in multiple BOINC projects, and can specify how their resources are allocated among these projects. BOINC provides a feature called homogeneous redundancy and a mechanism of credit and accounting to control erroneous computational results due to malfunctioning computers or to malicious participants. In this paper it is described the design issues, the goals of BOINC, and the solutions to the main problems.**

Index Terms—**Public-resource computing, homogeneous redundancy**

I. Public-Resource Computing

The worlds computing power and disk space is no longer primarily concentrated in supercomputer centers and machine rooms. Instead it is distributed in hundreds of millions of personal computers and game consoles belonging to the general public. Public-resource computing (also known as Global Computing or Peer-to-peer computing) uses these resources to do scientific supercomputing. This paradigm enables previously infeasible research. It also encourages public awareness of current scientific research, it catalyzes global communities centered around scientific interests, and it gives the public a measure of control over the directions of scientific progress. The idea of using the millions of fast computers, connected by the network, 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. In 1999 a third projects, seti@home [1], 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. SETI@home has attracted millions of participants worldwide. The number of Internet-connected PCs is growing rapidly, and is projected to reach 1 billion by 2015. Together, these PCs could provide many PetaFLOPs of computing power. The public resource approach applies to storage as well as computing. If 100 million computer users each provide 10 Gigabytes of storage, the total (one Exabyte, or 1018 bytes) would exceed the capacity of any centralized storage system. In spite of this resource, and an abundance of promising applications, relatively few large-scale public-resource projects have emerged. This is due in part to the lack of appropriate middleware (client and server software, management tools, user-centered web features, and so on). Some open-source systems have been developed, such as Cosm, jxta, and XtremWeb [2], but these systems provide only part of the necessary functionality. Commercial systems such as Entropia [3] and United Devices are more full-featured but not free.

II. BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) is a platform for public-resource distributed computing. BOINC is being developed at U.C. Berkeley Spaces Sciences Laboratory by the group that developed and continues to operate SETI@home. BOINC is open source and is available at http://boinc.berkeley.edu. It is being used for applications in physics, molecular biology, medicine, chemistry, astronomy, climate dynamics, mathematics, and the study of games. There are currently about 40 BOINCbased projects and about 400,000 volunteer computers performing an average of over 400 TeraFLOPS. BOINC projects are independent; each has its own database of jobs and volunteer accounts. Job durations vary widely between projects, ranging from a few minutes to several months (for example, Climateprediction.net [4]). Volunteers participate by running BOINC client software on their computers (hosts). BOINC is open source and the client is available for most platforms, including Windows, Linux, and Mac OS X. Volunteers can attach each host to any set of projects, and can specify various preferences that constrain when and how their resources are used. For example, they can control the allocation of resources among projects. There are advantages in attaching hosts to multiple projects. First, a given project may have periods when it has no work, so a host attached to several projects is less likely to become idle. Second, such a host may be able to do use its resource more fully, for example by downloading files for one project while computing for another.

III. The BOINC architecture

BOINC consists of client and server components (see Figure 1). The BOINC client runs projects' applications. The applications are linked with a runtime system whose functions include process control, checkpoint control, and graphics [5]. The client performs CPU scheduling (implemented on top of the local operating system's scheduler; at the OS level, BOINC runs applications at zero priority). It may preempt applications either by suspending them (and leaving them in memory) or by instructing them to quit. All network communication in BOINC is initiated by the client. A client communicates with a project's task server [6] via HTTP. The request is an XML document that includes a description of the host hardware and availability, a list of completed jobs, and a request for a certain amount (expressed in terms of CPU time) of additional work. The reply message includes a list of new jobs (each described by an XML element that lists the application, input and output files, including a set of data servers from which each file can be downloaded). Some hosts have intermittent physical network connections (for example, portable computers or those with modem connections). Such computers may connect only every few days. During a period of network connection, BOINC attempts to download enough work to keep the computer busy until the next connection.

A. The BOINC server

BOINC-based projects are autonomous. Each project operates a server consisting of several components:

- Web interfaces for account and team management, message boards, and other features.
- A task server that creates tasks, dispatches them to clients, and processes returned tasks.
- A data server that downloads input files and executables, and that uploads output files.

These components share various data stored on disk, including relational databases and upload/download files(see Figure 1).

B. The BOINC client

The BOINC client software consists of several components (see Figure 1):

- Applications are typically long-running scientific programs. They may consist of a single process or a dynamic set of multiple processes.
- The BOINC core client program communicates with schedulers, uploads and downloads files, and executes and coordinates applications.
- The BOINC Manager provides a graphical interface allowing users to view and control computation status. For each task, it shows the fraction done and the estimated time to completion, and lets the user open a window showing the applications graphics. It communicates with the core client using remote procedure calls over TCP.
- A BOINC screensaver (if enabled by the volunteer) runs when the computer is idle. It does not generate screensaver graphics itself, but rather communicates with the core client, requesting that one of the running applications display full-screen graphics.

The core client schedules applications. On a multiprocessor with n CPUs, it attempts (if user preferences allow it) to run n applications at once. It does preemptive round robin scheduling among applications, so that volunteers who participate in multiple projects see periodic change. Applications that are preempted may be suspended or forced to exit, depending on volunteer preferences. The core client also guards against certain types of application misbehavior. Each task has project-specified limits on memory usage, disk usage, and computation (number of floating-point operations). The runtime system periodically measures these quantities and reports them to the core client. If the limits are exceeded, the core client aborts the application. The core client interacts with applications in various ways. This interaction is implemented by a runtime system implemented by the core client and a runtime library linked with applications. Most of this interaction is hidden from the application developer. In cases where the developer needs to be involved, this is done via a BOINC Application Programming Interface (API).

IV. Design issues and solutions

A. Describing computation and data

BOINC uses a simple but rich set of abstractions for files, applications, and data. A project defines application versions for various platforms (Windows, Linux/x86, Mac OS/X, etc.). An application can consist of an arbitrary set of files. A workunit represents the inputs to a computation: the application (but not a particular version) a set of references input files, and sets of command-line arguments and environment variables. Each workunit has parameters such as compute, memory and storage requirements and a soft deadline for completion. A result represents the result of a computation: it consists of a reference to a workunit and a list of references to output files. Files (associated with application versions, workunits, or results) have project-wide unique names and are immutable. Files can be replicated: the description of a file includes a list of URLs from which it may be downloaded or uploaded. Files can have associated attributes indicating, for example, that they should remain resident on a host after their initial use, that they must be validated with a digital signature, or that they must be compressed before network transfer. When the BOINC client communicates with a scheduling server it reports completed work, and receives an XML document describing a collection of the above entities. The client then downloads and uploads files and runs applications; it maximizes concurrency, using multiple CPUs when possible and overlapping communication and computation. BOINCs computational system also provides a distributed storage facility (of computational inputs or results, or of data not related to distributed computation) as a byproduct. This storage facility is much different from peer-to-peer storage systems such as Gnutella, PAST [7] and Oceanstore [8]. In these systems, files can be created by any peer, and there is no central database of file locations. This leads to a set of technical problems (e.g. naming and file location) that are not present in the BOINC facility.

B. Redundant computing

Public-resource computing projects must deal with erroneous computational results. These results arise from malfunctioning computers (typically induced by overclocking) and occasionally from malicious participants. BOINC provides support for redundant computing, a mechanism for identifying and rejecting erroneous results. A project can specify that N results should be created for each workunit. Once $M \leq N$ of these have been distributed and completed, an application-specific function is called to compare the results and possibly select a canonical result. If no consensus is found, or if results fail, BOINC creates new results for the workunit, and continues this process until either a maximum result count or a timeout limit is reached. Malicious participants can potentially game the system by obtaining large numbers of results and detecting groups of results that comprise a quorum. BOINC makes this difficult by a work-distribution policy that sends only at most one result of a given workunit to a given user. Projects can also limit the total number of results sent to a given host per day. BOINC implements redundant computing using several server daemon processes:

- The transitioner implements the redundant computing logic: it generates new results as needed and identifies error conditions.
- The validater examines sets of results and selects canonical results. It includes an application-specific result-comparison function.
- The assimilater handles newly-found canonical results. Includes an application-specific function which typically parses the result and inserts it into a science database.
- The file deleter deletes input and output files from data servers when they are no longer needed.

In this architecture servers and daemons can run on different hosts and can be replicated, so BOINC servers are scalable. Availability is enhanced because some daemons can run even while parts of the project are down (for example, the scheduler server and transitioner can operate even if the science database is down). Some numerical applications produce different outcomes for a given workunit depending on the machine architecture, operating system, compiler, and compiler flags. In such cases it may be difficult to distinguish between results that are correct but differ because of numerical variation, and results that are erroneous. BOINC provides a feature called homogeneous redundancy for such applications. When this feature is enabled, the BOINC scheduler send results for a given workunit only to hosts with the same operation system name and CPU vendor. In this case, strict equality can be used to compare results. BOINC is compatible with other schemes for ensuring result correctness [9].

C. Failure and backoff

Public-resource computing projects may have millions of participants and a relatively modest server complex. If all the participants simultaneously try to connect to the server, a disastrous overload condition will generally develop. BOINC has a number of mechanisms to prevent this. All client/server communication uses exponential backoff in the case of failure. Thus, if a BOINC server comes up after an extended outage, its connection rate will be the longterm average. The exponential backoff scheme is extended to computational errors as well. If, for some reason, an application fails immediately on a given host, the BOINC client will not repeatedly contact the server; instead, it will delay based on the number of failures.

D. Participant preferences

Computer owners generally participate in distributed computing projects only if they incur no significant inconvenience, cost, or risk by doing so. BOINC lets participants control how and when their resources are used. Using these controls, called general preferences, participants specify the hysteresis limits of work buffering on machines (which determines the frequency of network activity); whether BOINC can do work while mouse/keyboard input is active; during what hours can BOINC do work; how much disk space can BOINC use; how much network bandwidth can BOINC use; and so on. These preferences are edited via a web interface, and are propagated to all hosts attached to the account. Participants can create separate sets of preferences for computers at home, work, and school. Some non-obvious controls are important to certain classes of participants. For example, DSL service in some countries has monthly transfer limits (typically a few hundred MB). BOINC provides a preference for upload, download and combined transfer limits over arbitrary periods. Some BOINC-based applications perform computations that are so floating-point intensive that they cause CPU chips to overheat. BOINC allows users to specify a duty cycle for such applications on a given CPU.

E. Credit and accounting

BOINC provides an accounting system in which there is a single unit of ”credit”, a weighted combination of computation, storage, and network transfer. This can be metered in various ways. By default, the BOINC client runs benchmarks on each CPU, and a results ”claimed credit” is based on this benchmark and elapsed CPU time. Credit ”cheating” is made difficult using the redundancy mechanism described above: Each result claims a certain amount of credit, but is granted only the average or minimum (the policy is project-specificed) of the claimed credit of correct results. Our experience with SETI@home has shown that participants are highly motivated by credit, and are particularly interested in their ranking relative to other users. This information is typically displayed on web-based ”leaderboards” showing the ranking of participants or teams of participants. There are many ways in which leaderboards can be subdivided, filtered, ordered, and displayed. For example, a leaderboard might show only participants from a particular country, or only those using a single PC; and it might rank entities by total or recent average credit. Rather than supply all these views, BOINC provides a mechanism that exports credit-related data (at the level of participant, team, and host) in XML files that can be downloaded and processed by credit statistics sites operated by third parties. Several of these currently exist. As part of the accounting system, BOINC provides a cross-project identification mechanism that allows accounts on different projects with the same email address to identified, in a way that does not allow email addresses to be inferred. This mechanism allows leaderboard sites to display credit statistics summed over multiple BOINC-based projects. Participants demand immediate gratification; they want to see their credit totals increase at least daily. Thus projects with long workunits (such as the climate prediction projects) need to grant credit incrementally as the workunit is being processed. BOINC offers a trickle messages mechanism, providing bidirectional, asynchronous, reliable, ordered messages, piggybacked onto the regular client/server RPC traffic. This can be used to convey credit or to report a summary of computational state; in the latter case, reply messages can abort wayward computations.

F. User community features

BOINC provides participant-oriented web sites features such as the ability to form teams, the ability to create and browse "user profiles" including text and images and message boards, including a dynamic FAQ system that encourages participants to answer each others questions. These facilities are integrated with the accounting system: credit and seniority provide a form of reputation system [10]. These features are important in attracting and retaining participants, and in providing a "customer support" mechanism that consumes little project resources.

G. Handling large numbers of platforms

Although the bulk of public computing resources use the Windows/Intel platform, there are many other platforms, far more than can easily be accessed by a typical project. BOINC provides a flexible and novel framework for distributing application executables. Normally, these are compiled and distributed by the project itself, for a given set of platforms (those accessible to the project). This mechanism is fine for most participants, but its inadequate for participants who, for security reasons, want to only run executables they have compiled themselves, participants whose computers have platforms not supported by the project and participants who want to optimize applications for particular architectures. To meet these needs BOINC provides an anonymous platform mechanism, usable with projects that make their application source code available. Participants can download and compile the application source code (or obtain executables from a third-party source) and, via an XML configuration file, inform the BOINC client of these application versions. Then, when the client communicates with that projects server, it indicates that its platform is "anonymous" and supplies a list of available application versions; the server supplies workunits (but not application versions) accordingly.

H. Local scheduling

The BOINC core client, in its decisions of when to get work and from what project, and what tasks to execute at a given point, implements a local scheduling policy. This policy has several goals:

- To maximize resource usage (i.e. to keep all processors busy);
- To satisfy result deadlines;
- To respect the participants resource share allocation among projects;
- To maintain a minimal variety among projects. This goal stems from user perception in the presence of long workunits.

Participants will become bored or confused if they have registered for several projects and see only one project running for several months. The core client implements a scheduling policy, based on a dynamic "resource debt" to each project, that is guided by these goals.

V. Projects using BOINC

A number of public-resource computing projects are using BOINC. The requirements of these projects have shaped the design of BOINC.

SETI@home A continuation of the original SETI@home project [1], performs digital signal processing of radio telescope data from the Arecibo radio observatory. A BOINC-based version of this project has been developed, and we are currently shifting the existing SETI@home user base (over 500,000 active participants) to the BOINC-based version. The BOINCbased SETI@home will use client disks to archive data, eliminating the need for its central tape archive.

Predictor@home [11] This project, based at The Scripps Research Institute, studies protein behavior using CHARMM, a FORTRAN program for macromolecular dynamics and mechanics. It is operational within Scripps, and is being readied for a public launch.

Folding@home [12] This project is based at Stanford University. It studies protein folding, misfolding, aggregation, and related diseases. It uses novel computational methods and distributed computing to simulate time scales thousands to millions of times longer than previously achieved. A BOINC-based project has been implemented and is being tested.

Climateprediction.net [4]The aim of this project (based at Oxford University) is to quantify and reduce the uncertainties in long-term climate prediction based on computer simulations. This is accomplished by running large numbers of simulations with varying forcing scenarios (initial and boundary conditions, including natural and manmade components) and internal model parameters. The Climateprediction.net application (a million-line FORTRAN program) produces a 2 GB detailed output file for each 50-year simulation run (which takes about 3 PC-months). These output files need to be uploaded and examined in a small fraction of cases - for example, when the smaller summary output file indicates a possible bug in the model.

Climate@home. This project is a collaboration of researchers at NCAR, MIT, UCAR, Rutgers, Lawrence Berkeley Lab, and U.C. Berkeley. Its scientific goals are similar to those of Climateprediction.net, but it will be using the NCAR Community Climate System Model (CCSM). It will collaborate with Climateprediction.net to maximize compatibility and minimize redundant effort, and to enable a systematic comparison of different climate models.

CERN projects. CERN (in Geneva, Switzerland) is deploying a BOINC-based project on 1,000 in-house PCs. The projects current application is a FORTRAN program that simulates the behavior of the LHC (Large Hadron Collider) as a function of the parameters of individual superconducting magnets. CERN researchers are investigating other applications.

Einstein@home This project involves researchers from University of Wisconsin, U.C. Berkeley, California Institute of Technology, LIGO Hanford Observatory, University of Glasgow, and the Albert Einstein Institute. Its purpose is to detect certain types of gravitational waves, such as those from spinning neutron stars, that can be detected only by using highly selective filtering techniques that require extreme computing power. It will analyze data from the Laser Interferometry Gravitational Observatory (LIGO) and the British/German GEO6000 gravitational wave detector.

UCB/Intel study of Internet resources. This project, a collaboration between researchers at the U.C. Berkeley Computer Sciences Department and the Intel Berkeley Research Laboratory, seeks to study the structure and performance of the consumer Internet, together with the performance, dependability and usage characteristics of home PCs, in an effort to understand what resources are available for peer-to-peer services. This project need to perform actions at specific times of day, or in certain time ranges. While performing these actions other BOINC applications must be suspended. The BOINC API supports these requirements.

VI. Contrast with Grid computing

Public-resource computing and Grid computing share the goal of better utilizing existing computing resources. However, there are profound differences between the two paradigms, and it is unlikely that current Grid middleware [13] will be suitable for public-resource computing. Grid computing involves organizationally-owned resources: supercomputers, clusters, and PCs owned by universities, research labs, and companies. These resources are centrally managed by IT professionals, are powered on most of the time, and are connected by full-time, high-bandwidth network links. There is a symmetric relationship between organizations: each one can either provide or use resources. Malicious behavior such as intentional falsification of results would be handled outside the system, e.g. by firing the perpetrator. In contrast, public resource computing involves an asymmetric relationship between projects and participants. Projects are typically small academic research groups with limited computer expertise and manpower. Most participants are individuals who own Windows, Macintosh and Linux PCs, connected to the Internet by telephone or cable modems or DSL, and often behind network-address translators (NATs) or firewalls. The computers are frequently turned off or disconnected from the Internet. Participants are not computer experts, and participate in a project only if they are interested in it and receive "incentives" such as credit and screensaver graphics. Projects have no control over participants, and cannot prevent malicious behavior. Accordingly there are different requirements on middleware for public resource computing than for Grid computing. For example, BOINCs features such as redundant computing, cheat-resistant accounting, and support for user-configurable application graphics are not necessary in a Grid system. Conversely, Grid computing has many requirements that public-resource computing does not. A Grid architecture must accommodate many existing commercial and research-oriented academic systems, and must provide a general mechanism for resource discovery and access. In fact, it must address all the issues of dynamic heterogeneous distributed systems, an active area of Computer Science research for several decades. This has led to architecture such as Open Grid Services Architecture [14], which achieve generality at the price of complexity and, to some extent, performance.

VII. Conclusion

It is described the public-resource computing paradigm and presented the design of a software system, BOINC, that facilitates it. BOINCs general goal is to advance the public resource computing paradigm: to encourage the creation of many projects, and to encourage a large fraction of the worlds computer owners to participate in one or more projects. BOINC reduces the barriers of entry to public-resource computing; in fact it allows a research scientist with moderate computer skills to create and operate a large public-resource computing project with about a week of initial work and an hour per week of maintenance. The server for a BOINC based project can consist of a single machine configured with common open-source software (Linux, Apache, PHP, MySQL, Python). BOINC shares resources among autonomous projects. Projects are not centrally authorized or registered. Each project operates its own servers and stands completely on its own. Nevertheless, PC owners can seamlessly participate in multiple projects, and can assign to each project a "resource share" determining how scarce resource (such as CPU and disk space) are divided among projects. If most participants register with multiple projects, then overall resource utilization is improved: while one project is closed for repairs, other projects temporarily inherit its computing power. On a particular computer, the CPU might work for one project while the network is transferring files for another. BOINC supports diverse applications, it provides flexible and scalable mechanism for distributing data, and its scheduling algorithms intelligently match requirements with resources. Existing applications in common languages (C, C++, FORTRAN) can run as BOINC applications with little or no modification. An application can consist of several files (e.g. multiple programs and a coordinating script). New versions of applications can be deployed with no participant involvement. Finally public-resource computing projects must provide "incentives" in order to attract and retain participants. The primary incentive for many participants is credit: a numeric measure of how much computation they have contributed. BOINC provides a credit accounting system that reflects usage of multiple resource types (CPU, network, disk), is common across multiple projects, and is highly resistant to ”cheating” (attempts to gain undeserved credit). BOINC also makes it easy for projects to add visualization graphics to their applications, which can provide screensaver graphics.

References

[1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, “Seti@home: An experiment in public-resource computing,” *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, November 2002.

[2] G. Fedak, C. Germain, V. Neri, and F. Cappello, “Xtremweb: A generic global computing platform,” *IEEE/ACM - CCGRID’2001 Special Session Global Computing on Personal Devices.*

[3] A. Chien, B. Calder, S. Elbert, and K. Bhatia, “Entropia: architecture and performance of an enterprise desktop grid system,” *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 597–610, May 2003.

[4] G. Fedak, C. Germain, V. Neri, and F. Cappello, “Climateprediction.net: Design principles for public-resource modeling research,” *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing Systems.*

[5] D. P. Anderson, C. Christensen, and B.Allen, “Designing a runtime system for volunteer computing,” *to appear in Supercomputing 06.*

[6] D. P. Anderson, E. Korpela, and R. Walton, “High-performance task distribution for volunteer computing,” *1st IEEE International Conference on e-Science and Grid Computing, Melbourne.*

[7] A. Rowstron and P. Druschel, “Storage management and caching in past, a large-scale persisten peer-to-peer storage facility.” *Symposium on Operating System Principles.*

[8] J. Kubiatowicz, D. Bindel, Y. Chen, P.Eaton, D. Geels, R.Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, “Oceanstore: An architecture for global-scale persistent storage.” *Proceedings of CAM ASPLOS.*

[9] C. Germain, “Result checking in global computing systems,” *ACM Int. Conf. on Supercomputing (ICS 03).*

[10] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, “Reputation systems,” *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, December 2000.

[11] P. http://predictor.scripps.edu.

[12] V. Pande and all, “Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing,” *Biopolymers*, vol. 68.

[13] I. Foster and C. Kesselman, “Globus: A metacomputing infrastructure toolkit,” *Int’l Supercomputer Applications*, vol. 11, no. 2.

[14] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, “Grid services for distributed systems integration,” *IEEE Computer*, vol. 35, no. 6.

Linda Ponta was born in Tortona (AL) in 1979. She received the master degree in Electronic Engineering with honours from the University of Genoa, Genoa, Italy, in March 2004. In the same year she has collaborated with the CINEF group at the University of Genoa. In January 2005 she has started the PhD in Electronic and Computer Science Engineering. Her primary research interests are in the field of agent-based simulation and data analysis.

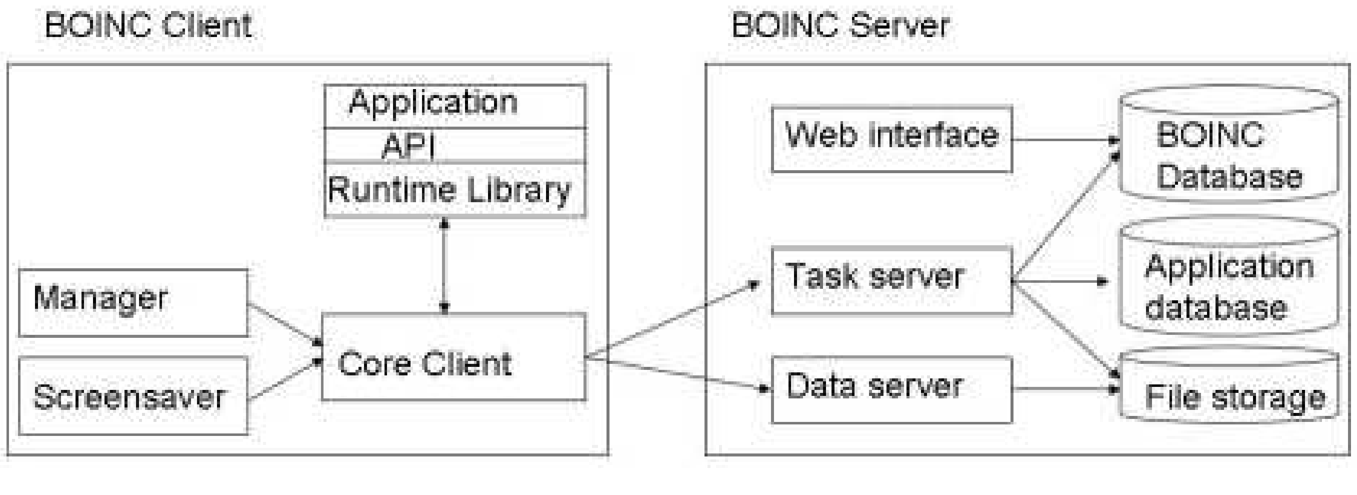

Fig. 1. Boinc Architecture