Volunteer Computing

Akhil Varupula, Kiran Panjam

April 21, 2020

Abstract

Our research on Volunteer computing is a very appealing way of utilizing vast available resources in an efficient way. Volunteer computing is a form of distributed computing, which allows public participants to share their idle computing resources, and helps run computationally expensive projects. Existing volunteer computing platforms consist of millions of users, providing huge amount of memory and processor for computation. Since there is a lot of growth in the volunteer computing because of the demand it posses for the resources, many researchers and organisations are coming forward to study and improve the current volunteer computing standards. The main purpose of our research paper to study the aspects, strengths, uses and requirements of the current standard of volunteer computing. Our paper also analyses the various projects like BOINC which are using volunteer computing as their platform to perform research. We also performed various surveys to enhance the working of volunteer computing and fill some gaps in the current volunteer computing research.

1 INTRODUCTION

In the earlier days in the countryside villages of old in the Philippines, when a family moved to a new place, they literally moved their whole house with them. First, they would place long bamboo poles under their house. Once this framework was in place, their friends and neighbours would then gather in numbers under it, and carry the house on their shoulders to its new location. This tradition, called Bayanihan, has been a favourite subject of artists and is still practiced today in some rural areas. More than a way to move houses, Bayanihan has come to be a dramatic symbol of the power of cooperation. Like its counterparts in other cultures, including Harambee in Africa and Barn raising in the United States, Bayanihan reminds us that seemingly impossible feats can become possible through the concerted effort of many people with a common goal and a sense of unity. Volunteer computing is primarily originated from this idea of Bayanihan.

Volunteer computing is type of distributed computing "an arrangement in which people, so-called volunteers, provide computing resources to projects, which use the resources to do distributed computing and/or storage". Thus, computer owners or users donate their computing resources (such as processing power and storage) to one or more “projects.” Volunteers are typically members of the general public who own Internet-connected personal computers. Organisations such as schools and businesses may also volunteer the use of their computers. Volunteers are effectively anonymous; although they may be required to register and supply email address or other information, they are not linked to a real-world identity.

There are currently about 30 VC projects in many scientific areas and at many institutions. About 700,000 devices are actively participating in Volunteer Computing projects. These devices have about 4 million CPU cores and 560,000 GPUs, and collectively provide an average throughput of 93 PetaFLOPS. The devices are primarily modern, high-end computers, they average 16.5 CPU GigaFLOPS and 11.4 GB of RAM, and most have GPU capable of general-purpose computing using OpenCL or CUDA.The potential capacity of VC is much larger: there are more than 2 billion consumer desktop and laptop computers. The monetary cost of this Vol-

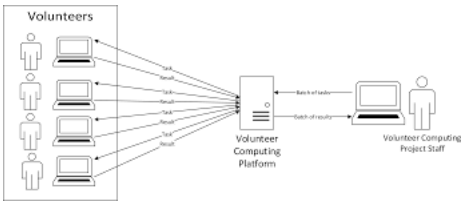

Figure 1:

unteer Computing project is divided between volunteers and scientists. Volunteers pay for buying and maintaining computing devices, for the electricity to power these devices, and for Internet connectivity. Scientists pay for a server and for the system administrators, programmers and web developers needed to operate the Volunteer Computing project.

1.1 EVOLUTION OF VOLUNTEER COMPUTING

Volunteer-computing systems were developed since half of 90-s, as answer to novel possibilities given by the rapidly expanding worldwide Internet at this time. First of such approaches to distributed computing was The Great Internet Mersenne Prime Search (GIMPS), that began in 1996, and has probably been the first project that used volunteer computing. GIMPS is still active and has already discovered fifteen new Mersenne prime numbers. The distributed.net project was founded in 1997 in the response to the RSA Secret-Key Challenge contest organized by the RSA Laboratories. Until now the project has managed to find two of cryptographic keys used in the contest. Volunteer whose device finds the key is given a monetary reward, which is an interesting way of attracting volunteers . The SETI@home (Search for Extra-Terrestrial Intelligence at Home) project has played a significant role in popularizing the idea of volunteer computing. The project uses volunteer computing to analyze signals from the Arecibo radio telescope to find signs of extra-terrestrial life. During the first week of existence around 200 000 people downloaded and run the client to become volunteers.

BOINC is a generic volunteer computing platform that can be used to create volunteer computing projects. Development of the BOINC (Berkeley Open Infrastructure for Network Computing) platform started in 2002 at the Space Sciences Laboratory at the University of California. The platform provides a consistent and easy to use interface where volunteers can choose projects they want to support and decide how much computing resources should be committed to each of them. Virtual credits are given to volunteers and their total numbers are published. Bayanihan project described in 1998 in the article titled "Bayani- han: Web-Based Volunteer Computing Using Java" was probably the first platform that allowed volunteers to use web browsers to take part in computations. In this platform, the means for computing were secured based on Java applets. From here Volunteer Computing is being used for bigger projects like BOINC(Berkeley Open Infrastructure for Network Computing) platform started in 2002 at the Space Sciences Laboratory at the University of California

2 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) is an 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. BOINC's general goal is to advance the public re- source computing paradigm: to encourage the creation of many projects, and to encourage a large fraction of the world's computer owners to participate in one or more projects. Specific goals include:

- **Reducing the barriers of entry to public-resource computing**: BOINC 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).

- **Share resources among autonomous projects**: BOINC-based projects are autonomous. 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 uti- lization 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.

- **Support diverse applications**: BOINC accommodates a wide range of applications; it provides flexible and scalable mechanism for distributing data, and its scheduling algorithms intelligently match requirements with resources. Existing applications in common languages can run as BOINC applications with little or no modification. An application can consist of several files. New versions of applications can be deployed with no participant involvement.

- **Reward participants**: Public-resource computing projects must provide "incentives" in order to attract and retain participants.he 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.

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 , 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 BOINC based SETI@home will use client disks to archive data, eliminating the need for its central tape archive.

- **Predictor@home,**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,**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,**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 and internal model parameters.

- **Climate@home,**This project is a collaboration of researchers at various universities. Its scientific goals are similar to those of Climateprediction.net, but it will be using the NCAR Community Climate System Model. 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, and plans to launch the project publicly in coordination with its 50th anniversary in October 2004. The project's current application is a FORTRAN program that simulates the behavior of the Large Hadron Collider as a function of the parameters of individual superconducting magnets.

- **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.

- **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.

A BOINC project corresponds to an organization or research group that does public-resource computing. It is identified by a single master URL, which is the home page of its web site and also serves as a directory of scheduling servers. Participants register with projects. A project can involve one or more applications, and the set of applications can change over time. The server complex of a BOINC project is centered around a relational database that stores descriptions of applications, platforms, versions, work units, results, accounts, teams, and many. Server functions are performed by a set of web services and daemon processes, Scheduling servers handles RPCs from clients; it issues work and handles reports of completed results. Data servers handles file uploads using a certificate-based mechanism to ensure that only legitimate files, with prescribed size limits, can be uploaded. File downloads are handled by plain HTTP. BOINC provides tools for creating, starting, stopping and querying projects; adding new applications, platforms, and application ver- sions, creating workunits, and monitoring server performance. BOINC is designed to be used by scientists, not system programmers or IT professionals; the tools are simple and well-documented, and a full-featured project can be created in a few hours. Participants join a BOINC-based project by visiting the project's web site, filling out a registration form, and downloading the BOINC client. The client can operate in several modes: as a screensaver that shows the graphics of running applications; as a Windows service, which runs even when no users are logged in and logs errors to a database; as an application that provides a tabular view of projects, work, file transfers, and disk usage, and as a UNIX command-line program that communicates through stdin, stdout and stderr, and can be run from a cron job or startup file. BOINC provides tools that let participants remotely install the client software on large numbers of machines, and attach the client to accounts on multiple projects.

- **The High-Level Structure of BOINC**: BOINC's architecture involves multiple components interacting through HTTP-based RPC interfaces. It is designed to be modular and extensible. Figure 2 shows the components of BOINC; these components are described subsequently. The shaded boxes represent the BOINC software distribution; unshaded boxes represent components developed by third parties.

2.1 Projects and Volunteers

In BOINC terminology, a project is an entity that uses BOINC to do computing. Each project has a server, based on the BOINC server software, that distributes jobs to volunteer devices. Projects are

autonomous and independent. A project may serve a single research group (SETI@home), a set of research groups using a common set of applications (Climateprediction.net, nanoHUB@home) or a set of unrelated scientists (IBM World Community Grid, BOINC@TACC). BOINC provides HTTP-based remote job submission APIs so that projects can submit and manage jobs using web-based interfaces (e.g., science gateways) or via existing systems such as HTCondor. Projects may operate a public-facing web site. The BOINC server software provides database-driven web scripts that support volunteer login, message boards, profiles, leader boards, badges, and other community and incentive functions. The project may extend this with their own web content, such as descriptions and news of their research. A project is identified by the URL of its web site. A volunteer is the owner of a computing device (desktop, laptop, tablet, or smartphone) who wants to participate in VC. They do so by
a) installing the BOINC client on the device
b) selecting projects to support and creating an account on each project, and
c) attaching the BOINC client to each account. Each attachment can be assigned a resource share indicating the relative amount of computing to be done, over the long term, for the project.

2.2 Client/Server Interaction

When attached to a project, the BOINC client periodically issues RPCs to the project's server to report completed jobs and get new jobs. The client then downloads application and input files, executes jobs, and uploads output files. All communication uses client-initiated HTTP; thus the BOINC client works behind firewalls that allow outgoing HTTP, and with HTTP proxies. The server must be publicly accessible via HTTP. Downloaded files can be compressed in transit; the client uncompresses them. Files need not be located on project servers. The validity of downloaded files is checked using hashes, and (for application files) code signing. Files are uploaded by an HTTP POST operation, which is handled by a CGI program on the server. Upload filenames include a random string to prevent spoofing. To prevent DoS

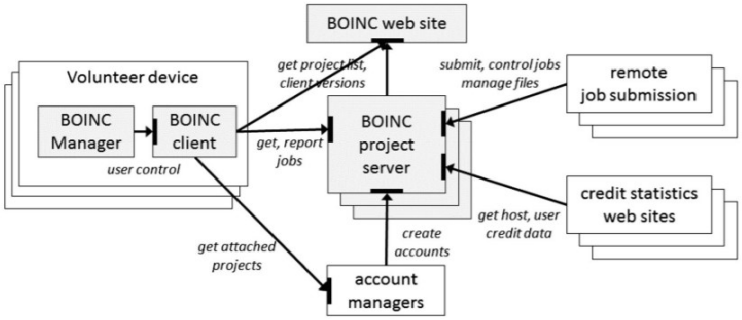

Figure 2:

attacks, projects can optionally use encryption-based tokens to limit the size of uploaded files. File uploads need not go to the project server; for example, Climateprediction.net collaborates with scientists in different countries, and the result files are sent directly to the collaborators' servers. The client also interacts occasionally with a server operated by the BOINC project itself, to obtain current lists of projects and of client versions. All client/server interactions handle failure using exponential back-off in order to limit the rate of requests when a server resumes after a period of being off-line.

3 Job Processing Abstractions and Features

3.1 Handling Resource Heterogeneity

The pool of volunteered computers is highly diverse. At a hardware level, computers may have Intelcompatible, Alpha, MIPS, or ARM CPUs, and their 32- and 64-bit variants. Intel-compatible CPUs can have various special instructions like SSE3. ARM processors may have NEON or VFP floating-point units. The computers run various operating systems (Windows, Mac OS X, Linux, FreeBSD, Android)

and various versions of these systems. There are a range of GPUs made by NVIDIA, AMD, and Intel, with various driver versions offering different features. BOINC provides a framework that lets projects use as many of these computing resources as possible, and to use them as efficiently as possible. Projects can build versions of their programs for specific combinations of hardware and software capabilities. These are called app versions, and the collection of them for a particular program is called an app. Jobs are submitted to apps, not app versions. BOINC defines a set of platforms, which are generally the combination of a processor type and operating system: for example, (Windows, Intelx64). Each app version is associated with a platform, and is sent only to computers that support that platform. For finer-grained control of version selection, BOINC provides the plan class mechanism. A plan class is a function that takes as input a description of a computer (hardware and software), and returns a) whether an app version can run on that computer; b) if so what resources (CPU and GPU, possibly fractional) it will use, and c) the peak FLOPS of those resources. For example, a plan class could specify that an app version requires a particular set of GPU models, with a minimum driver version. Plan classes can be specified either in XML or as C++ functions. App versions can optionally be associated with a plan class. App versions have integer version numbers. In general, new jobs are processed using the latest version for a given (platform, plan class) pair; in-progress jobs may be using older versions. When the BOINC client requests jobs from a server, it includes a list of platforms it supports, as well as a description of its hardware and software. When the server dispatches a job to a computer, it selects an app version with a matching platform, and whose plan class function accepts the computer.

3.2 Jobs Properties

A BOINC job includes a reference to an app and a collection of input files. A job has one or more instances. Jobs have input files; instances have output files. When the server dispatches an instance to a client, it specifies a particular app version to use. The client downloads the input files and the files that make up the app version, executes the main program, and (if it completes successfully) uploads the resulting output files and the standard error output. In the following we use "FLOPs" (lower-case s) as the plural of FLOP (floating-point operation). FLOPS (upper-case s) denotes FLOPs per second. Each job has a number of properties supplied by the submitter. These include:
a) An estimate of the number of FLOPs performed by the job. This is used to predict runtime;
b) A maximum number of FLOPs, used to abort jobs that go into an infinite loop.
c) An estimate of RAM working-set size, used in server job selection.
d) An upper bound on disk usage, used both for server job selection and to abort buggy jobs that use unbounded disk space.
e) An optional set of keywords describing the job's science area and origin.

3.3 Result Validation

Hosts may return incorrect results because of hardware errors or malicious user behavior. Projects typically require that the final results of the job are correct with high probability. There may be application-specific ways of doing this: for example, for physical simulations one could check conservation of energy and the stability of the final configuration. For other cases, BOINC provides a mechanism called replication-based validation in which each job is processed on two unrelated computers. If the outputs agree, they are accepted as correct; otherwise a third instance is created and run. BOINC: A Platform for Volunteer Computing This is repeated until either a quorum of consistent instances is achieved, or a limit on the number of instances is reached. What does it mean for two results to agree? Because of differences in floating-point hardware and math libraries, two computers may return different but equally valid results. By carefully selecting compilers, compiler options, and libraries it's possible to get bitwise-identical results for FORTRAN programs across the major platforms. In general, however, discrepancies will exist. For applications that are numerically sta-

ble, these discrepancies lead to small differences in the final results. BOINC allows projects to supply application-specific validator functions that decide if two results are equivalent, i.e. whether corresponding values agree within specified tolerances. Physical simulations are typically not stable. For such applications, BOINC provides a mechanism called homogeneous redundancy in which computers are divided into equivalence classes, based on their hardware and software, such that computers in the same class compute identical results for the app. Once a job instance has been dispatched to a computer, further instances are dispatched only to computers in the same equivalence class. BOINC supplies two equivalence relations: a coarse one involving only OS and CPU vendor, and a finer-grained one involving CPU model as well. Projects can define their own equivalence relations if needed. In some cases there are discrepancies between app versions; for example, CPU and GPU versions may compute valid but incomparable results. To address this, BOINC provides an option called homogeneous app version. Once an instance has been dispatched using a particular app version, further instances use only the same app version. This can be used in combination with homogeneous redundancy. Basic replication-based validation reduces effective computing capacity by a factor of at least two. BOINC provides a refinement called adaptive replication that moves this factor close to one. The idea is to identify hosts that consistently compute correct results (typically the vast majority of hosts) and use replication only occasionally for jobs sent to these hosts. Actually, since some computers are reliable for CPU jobs but unreliable for GPU jobs, we maintain this "reputation" at the granularity of (host, app version). Adaptive replication works as follow: the BOINC server maintains, for each (host, app version) pair, a count N of the number of consecutive jobs that were validated by replication. Once N exceeds a threshold, jobs sent to that host with that app version are replicated only some of the time; the probability of replication goes to zero as N increases. Adaptive replication can achieve a low bound on the error rate (incorrect results accepted as correct), even in the presence of malicious volunteers, while imposing only a small throughput overhead. Result validation is useful for preventing credit cheating. Credit is granted only to validated results, and "faked" results generally will fail validation. For applications that usually return the same answer (e.g. primality checks), cheaters could simply always return this result, and would usually get credit. To prevent this, the application can add extra output that depends on intermediate results of the calculations

4 The Life of a Job

In most batch systems the life of a job is simple: it gets dispatched to a node and executed. In BOINC, however, a dispatched job has various possible outcomes:

1. The job completes successfully and correctly.
2. The program crashes.
3. The job completes but the returned results are incorrect, either because of hardware malfunction or because a malicious volunteer intentionally substitutes wrong results.
4. The job is never returned, because (for example) the computer stops running BOINC. To ensure the eventual completion of a job, and to validate its results, it may be necessary to create additional instances of the job. Each job J is assigned a parameter delay_bound(J) by its submitter. If an instance of J is dispatched to a host at time T, the deadline for its completion is T + delay_bound(J). If the completed results have not been returned to the server by the deadline, the instance is assumed to be lost (or excessively late) and a new instance of J is created. The scheduler tries to dispatch jobs only to hosts that are likely, based on runtime estimates and existing queue sizes, to complete them by their deadlines. Projects may have time-sensitive workloads: e.g., batches of jobs that should be finished quickly because the submitter needs the results to proceed. Such jobs may be assigned low delay bounds; however, this may limit the set of hosts to which the job can be dispatched, which may have the opposite of the desired effect. For workloads that are purely throughput-oriented, delay_bound(J) can be large,

allowing even slow hosts to successfully finish jobs. However, it can't be arbitrarily large: while the job is in progress, its input and output files take up space on the server, and delay_bound(J) limits this period. For such workloads, a delay bound of a few weeks may be appropriate. In addition to delay_bound(J), a job has the following parameters; these are assigned by the submitter, typically at the level of app rather than job: & min_quorum(J): validation is done when this many successful instances have completed. If a strict majority of these instance have equivalent results, one of them is selected as the canonical instance, and that instance is considered the correct result of the job. & init_ninstances(J): the number of instances to create initially. This must be at least min_quorum(J); it may be more in order to achieve a quorum faster. & max_error_instances(J): if the number of failed instances exceeds this, the job as a whole fails and no more instances are created. This protects against jobs that crash the application. & max_success_instances(J): if the number of successful instances exceeds this and no canonical instance has been found, the job fails. This protects against jobs that produce nondeterministic results. So the life of a job J is as follows: & J is submitted, and init_ninstances(J) instances are created and eventually dispatched. & When an instance's deadline is reached and it hasn't been reported, the server creates a new instance. & When a successful instance is reported and there is already a canonical instance, the new instance is validated against it to determine whether to grant credit. Otherwise, if there are at least min_quorum(J) successful instances, the validation process is done on these instances. If a canonical instance is found, the app's assimilator is invoked to handle it. If not, and the number of successful instances exceeds max_success_instances(J), J is marked as failing. & When a failed instance is reported, and the number of failed instances exceeds max_error_instances(J), J is marked as failing. Otherwise a new instance is created. When a canonical instance is found, J's input files and the output files of other instances can be deleted, and any unsent instances are cancelled. BOINC: A Platform for Volunteer Computing. It's possible that a successful instance of J is reported after a canonical instance is found. It's important to grant credit to the volunteer for the instance, but only if it's correct. Hence the output files of the canonical instance are retained until all instances are resolved. After all instances are resolved, and the job has succeeded or failed, the job and job instance records can be purged from the database. This keeps the database from growing without limit over time; it serves as a cache of jobs in progress, not an archive of all jobs. However, the purging is typically delayed for a few days so that

4.1 Requirements for volunteer computing platforms

Generally, VC follows a master-worker parallel computing model as shown in Figure 3 . In this model the master disintegrates massive tasks into small chunks. The master then distributes these small chunks among workers. Workers perform required computation and send results back to the master. The master then verifies data results and aggregates them to compute final results. All operations of a VC system are handled by a middleware. A VC system must ensure the following requirements:
(i) efficient division of tasks into small chunks, so that they may be resourcefully executed over workers,
(ii) robust scheduling of tasks and resources, so that the overall throughput of the system will increase,
(iii) modeling communication and computation of huge number of sporadic resources, (iv) trust: the volunteers trust middleware of different projects that do not harm their systems or attack their privacy,
(v) truthfulness: the volunteers trust that the project is truthful about the claimed applications of project, claimed work done by their middleware, and how the processing and storage resources (intellectual property) will be used, and
(vi) security: the volunteer trusts the projects to use appropriate security policies, so that hackers may not use the project as a source for malicious activities (Volunteer computing, 2013; Security Threats, 2012).
Currently, BOINC is the most popular platform. It is an opensource platform for VC. The BOINC model

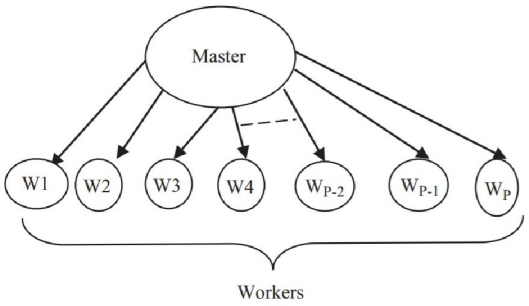

Figure 3:

involves volunteers and projects. Volunteers are users who participate by contributing their resources in the form of storage and processing. Projects are organizations (normally academic research groups) that need computational power. Each project runs on its own BOINC server. Volunteers contribute by running BOINC client on their computers and other devices. The popularity of BOINC stems in making it possible for scientists and researchers to hit into the massive processing power of heterogeneous devices around the world. BOINC has been developed by a team based at the Space Sciences Laboratory, University of California, Berkeley. As a high performance parallel distributed computing platform, BOINC has about 8,374,328 hosts, 540,130 active hosts with 7.4 petaFLOPS average processing as of January 2013 (Volunteer computing 2013). As shown in Figure 3, a VC system consists of workers and server side middleware. Workers consist of a middleware and project specific execution code. Workunit are executed on the BOINC workers using the project specific execution code. A BOINC server comprises of a data server, scheduling server, database server, and BOINC daemon processes such as work generator (generating new workunits), feeder (reorganize the scheduler's database access), transitioner (as needed, it generates new results and identifies error conditions); validator (validating results); assimilator (handle newly canonical results), deletor (deletes those input and output files that are no longer needed) and database purger (removes jobs and instance database entries that are no longer needed)

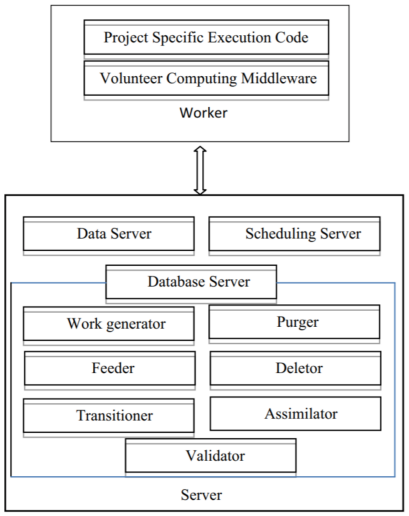

Figure 4:

4.2 Computation model of volunteer computing

This section discusses the computational model of VC. Generally, it consists of X sets of tasks denoted by S1,... Sx. Each set Si has N tasks JN. The submission time, computational time and deadline of each task i are represented by Subi, SiTi and Di respectively. Satisfied tasks are completed and reported before its deadline and its set is called a satisfied set. In VC systems, there are P workers W1...., WP to compute the tasks of a set. When the worker initially become available, it request for a connection. After specified intervals, a worker also requests for a connection. This interval is known as a reconnection time. A worker at any given time is either available or unavailable, whereas the project server is always available. If a worker is an uninterrupted available state, then it is called availability interval and a period where the worker is in uninterrupted unavailable state, then it is called unavailability interval. Hence, the lifespan of a worker is the interval between the start of the initial availability interval say AI and the end of the latest availability interval AL. In large scale distributed systems like VC, the key focus is the average availability, which is the fraction of worker lifespan spent in the available state. Through checkpointing, the task is resumed

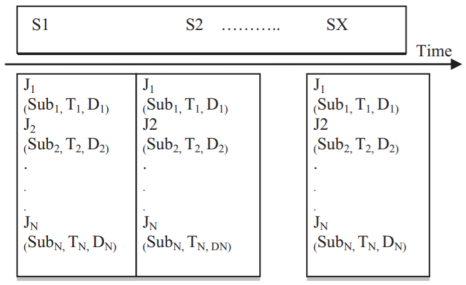

Figure 5:

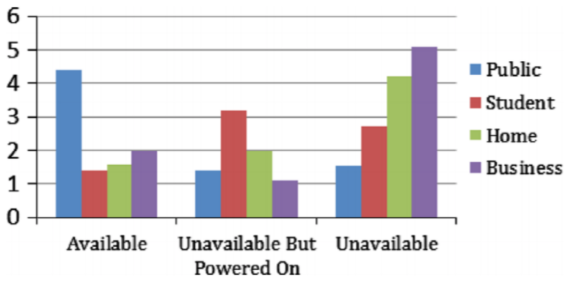

Figure 6:

at the same point, when a worker transitions from unavailable state to available state.

5 Research issues and challenges

In this section, main issues of existing VC are described. At the end some possible solutions described in the literature are also discussed. The foremost challenge in efficient utilization of hosts is that users have heterogeneous resources with varying capabilities. Proper distribution of tasks according to the capacity of each volunteer is desired in order to achieve timely completion of jobs. Further, this would ensure efficient utilization of resources, reduced cost of operation, and enhanced user satisfaction. Heterogeneity-aware distribution of workload would also permit priority-based execution of tasks and improve real time behavior. In addition, it may increase the pool of applications that may be executed on the VC platform. In order to efficiently utilize the huge number of volunteered resources, workers analysis, efficient size of a workunit, workers communication, computation, task distribution and results verification are important.

5.1 Analysis of volunteer computing workers

The workers traces allow us to know whether a worker is available when a powered-on worker is unavailable to work, and when a worker's computer is powered off. To accomplish effective task distribution, workers analysis is the foremost challenge. Specifically, the following issues are discussed in this section:
(i) How to measure CPU availability across millions of workers?
(ii) How to identify patterns of availability among voluntarily dedicated resources?
(iii) How to improve resource management in VC by applying knowledge of availability patterns?and
(iv) How to measure CPU availability across millions of workers? Middleware such as BOINC, xtremWeb, Xgrid, Grid MP are mainly used for gathering and measuring data at large-scale. The BOINC client is normally used for measuring CPU availability. It records the start and stop interval of CPU availability, and confine the temporal structure of CPU. There is a service to query the operating system every 10 s, to determine whether the screensaver was running or not. After a collection period of 28 days, 140 traces were recorded out of which 68 (49%)came from public computer labs, 25 (18%) from students, 20 traces (14%) from home users and 27 (19%) from business users. These results are shown in Figure 6. In order to check the nature of workers, CPU, and patterns of availability among volunteered resources, a modified BOINC client was distributed among 112,268 users. After a collection period of about seven months, the log files of these workers were collected at the BOINC server for SETI@home project. The logs collected during this time period traced 16,293 years of CPU availability. Through these logs files, the following observations were deduced:
Nature of workers: about 81% of the workers were at home, 17% at offices, and 2% at schools.

CPU availability: many P2P availability studies focus on the host availability but not on CPU. It has been observed, that resources may have 100% host availability but 0% CPU availability.
Volatility of resources: volatility of resources signify the friction of time they are available. It has been observed that 50% of workers are available for less than 40% and merely a small fraction of workers (about 4%) are available more than 85% of their lifespan. Hence, it makes difficult to use the existing scheduling algorithms, due to this downtime and highly unpredictable nature of workers Figure 6. Average period durations of workers. S1 S2 SX J1 (Sub1, T1, D1) J2 (Sub2, T2, D2) . . . JN (SubN, TN, DN) J1 (Sub1, T1, D1) J2 (Sub2, T2, D2) . . . JN (SubN, TN, DN) J1 (Sub1, T1, D1) J2 (Sub2, T2, D2) . . . JN (SubN, TN, DN) Time Figure 5. Computational model of VC having X sets S and each set have N jobs J. Submission, computational time T and deadline D of each job JN in a set SN may be different from each other. 0 50 100 150 200 250 300 350 Days Figure 5.
Workers lifespan: an analysis of VC projects shows that the lifespan of workers generally follows an exponential distribution with a mean of 90 days. The Mean interval length of 60% workers is less than 24 h, which suggests the requirement of fault-tolerance mechanism for extensively long-running applications. The mean and median lifespan of a worker is roughly 2–3 months. It means that tasks in length less than 15 h will fail less than 1% due to permanent worker unavailability. Group of resources with similar availability and
CPU speed: David et al. (Kondo et al, 2008) performed a series of experiments over these traces and have found the following results:
(i) nearly 10,000 workers having processing speed approximately 2 109 FPOPS are more than 90% available,
(ii) about 2500 workers having CPU frequency 2 109 are 30% available,
(iii) there is a little correlation between host speed and CPU availability,
(iv) Workers available more than 90% contribute 40% of the total CPU time,
(v) workers which are less available, contribute a lot, i.e. Workers having availability between 50% and 85%, contribute about 45%, and
(vi) dedicated workers with availability more than 95%, contribute less than 5% of their computational power.

5.2 Existing task distribution policies

The rapid growth of VC projects has attracted more researchers to study and improve the existing platform. The progress of concurrently running projects has slowed down due to the increasing competition for hosts. According to a recent report, there are 360 million internet connected users out of which less than 2% are participating in VC (Volunteer computing, 2013). Moreover, because of the high computational needs and low participation rate, it has become extremely important to attract more participants, and use their resources more efficiently. In the context of task scheduling, the quality of a schedule is guaranteed in spite of a certain level of performance fluctuation, such as resource performance degradation and incorrect estimates of task completion times. The authors have addressed the task scheduling problem based on proactive reallocation scheme which enables output schedules to tolerate a certain degree of performance degradation in VC system. According to them, rescheduling occurs only if a new task-resource match improves the robustness without increasing the make-span. Furthermore, it has been pointed out that task retrieval policies might force the clients to complete the number of successful tasks. Hence, the second main issue is the use of efficient task retrieval policy. Young and Heien et al.proposed a communication model of task requests for VC workers. Task requests from VC workers are modeled as a Poisson process, and through the workers reconnection period (T), the request rate of the process is controlled. Hence, rather than scheduling tasks requests for individual workers, the entire worker pool is considered as a tunable stream.Each worker is given a reconnection time given by ReconnectðTÞ$\frac{1}{4}$ CurrentðTÞþT=Pactive dh5ÞTable 1 Workers attributes in volunteer

computing system. Worker attributes Description Processor type Type of processor OS Operating system Worker type Desktop user, mobile User, video game console No. of Cores (Cn) No. of cores CPU frequency (CPUs) Processing speed Cores free (Cf) Percentage of free cores CPU free (CPUf) Percentage of free CPU Memory (Ms) Memory of the system Free memory (Mf) Free available memory Storage (Sf) Free storage Availability session (Aws) Worker's availability since the last connection Timeout workunits(Tw) Number of time-out workunits Workunit completed session (Wcs) Count of workunits completed since last connection Total workunit completed (wct) Without error or timeout over WUI distributed Reliability (Rws) Worker/application WUI valid over WUI collected since the last worker connection Availability(Awp) Worker WUI completed without error or timeout over WUI distributed since worker joined the project Reliability(Rwp) Worker/application WUI valid over WUI since worker joined the project Lifespan Worker number of hours since the worker joined the project Average turnaround Worker average time to return a WUI for a given worker Acquired workunits Worker/application number of WUIs given to the worker in the last connection Average dedicated time Worker average time dedicated to VC project Workunits in progress Worker number of WUIs still in progress Workunit in progress Application average size of WUs in flops where Pactive are the recently active workers in the last 2T seconds. They found that the average time gap between connections of 3.76 closely matches to the expected time gap equal to 4 h/4025 workers$\frac{1}{4}$3.58 s. Moreover, according to this task distribution scheme, "the time gap between connections to the master can be modeled as an exponential distribution function (EDF)". Best results can be obtained when the minimum p-value$\frac{1}{4}$0.05 for Kolmogorov–Smirnov (KS) test is applied to all results of simulation. Based on the communication model of task requests for VC workers, different task retrieval policies have been proposed by the researchers. In VC projects most existing task distribution policies are based on heuristics and can be categorized in two modules: naive and knowledge-based. Naïve scheduling policies assign computational tasks without taking into account the history of the workers. Following are examples of naive task distribution policies:

(i) First-come-first-served (FCFS): in this policy workunits for computation are sent to any worker who requests. The main disadvantage of FCFS policy is assigning tasks to even those workers that have shown low availability and reliability in the past.

(ii) Locality scheduling policy: here tasks are preferably sent to workers who already have the necessary data to accomplish the work. Workers having issues of availability and reliability in this policy may degrade the overall performance of the system.

(iii) Random assignment: in this scheme workers are randomly selected and workunits are sent to them. As the selection in this policy is random, the performance of the system will fluctuate time to time. In order to get steady results, workers may be grouped according to their availability and reliability patterns, and then the task may be distributed among them randomly.