Coordinating Volunteer Computing

David P. Anderson

Spaces Sciences Laboratory

University of California, Berkeley

1 March 2019

**Abstract**: Volunteer computing (VC) lets people donate the unused capacity of their computing devices (desktop, laptop, mobile) to science research projects. It can provide Exa-scale high-throughput computing, and offers a scalable and sustainable alternative to organizational approaches. Since its inception in 2004, BOINC-based volunteer computing has been based on a “free-market” model in which scientists create and promote projects, and volunteers discover, evaluate, and choose from among these projects. This model has limited the adoption of volunteer computing.

To move beyond these limits, we propose a new model in which volunteers register for science areas rather than projects, and a central “coordinator” manages the global allocation of computing resources to scientists.

# 1 Introduction

“Volunteer computing” (VC) does high-throughput computing using consumer resources volunteered by their owners. MORE.

A platform for VC, such as BOINC, defines standard processes by which scientists get computing power, and by which volunteers learn of VC and make their resources available to scientists. These processes define a “model” for VC.

## 1.1 The free-market VC model

BOINC’s original model was based on these processes:

* Scientists get computing power by a) learning about BOINC, b) creating and operating a BOINC “project” comprising a web site and a job dispatcher, and c) recruiting volunteers by publicizing their project and by providing web content describing their research, scientific credentials, and past results.
* Volunteers discover VC via the publicity of a project P, which takes them to P’s web site. This directs them to download the client software from the BOINC web site. When the BOINC client starts up, the volunteer is shown a list of projects, from which they select P, thus “attaching” the device to P. The volunteer, perhaps at a later time, surveys the other available projects, visits their web sites, and may choose to attach to some of them. (The BOINC client lets volunteers attach to multiple projects and control the division of resources among them.)

The intention of this model was to create 1) to create a dynamic “ecosystem” of projects that compete for computing power by promoting themselves and their science, and 2) a population of volunteers that periodically evaluate the set of projects and make informed decisions, based on their personal values and opinions, about how to allocate their computing resources.

This model has had some success: there have been about 40 projects, many of them doing significant published research. However, only a few significant new projects have arisen in recent years. We think this is due to the model; in particular:

1. Creating and operating a VC project is harder than expected: it requires a combination of resource and skills (sysadmin, DB admin, web design, PR and outreach) that few academic research groups have.
2. For a research group, using VC is a risk. There's a substantial investment, with no guarantee of any return, since no one may volunteer. Adding a VC component to a grant proposal adds uncertainty and weakens the proposal.
3. The computing needs of many research groups are sporadic; e.g. they need a big chunk of throughput every now and then. For such groups, buying computing time on a commercial cloud may be cheaper than using VC.

Similarly, although VC has attracted several hundred thousand volunteers, this population is slowly declining, it consists primarily of tech-savvy males, and most volunteers are “locked in” to a few projects and don’t actively seek out new ones. Again, the underlying issues are inherent in the model:

1. The complexity and technical jargon of the BOINC interface confuses and drives away many computer owners.
2. Attracting volunteers is a marketing problem. It's difficult to do effective marketing when there are dozens of competing brands (i.e. projects names).
3. Volunteers have little incentive to repeatedly survey and assess a large number of projects.

## 1.2 Account Managers

We recognized early on that it was inconvenient to browse lots of project web sites, and so we added a feature called the “account manager architecture”. An account manager (AM) is both a web site and an RPC server. Instead of attaching directly to projects, a volunteer can attach their device to an AM. The BOINC client periodically issues an RPC to the AM. The RPC reply contains a list of projects to which the client should attach.

This architecture was used by independent developers to create two AMs – Gridrepublic and BAM! – that allow volunteers to browse and select projects on a single web site. This eliminates the need to browse separate web sites in choosing projects, and it also makes it possible to efficiently manage the attachments of multiple devices. However, the AMs did not solve the basic problems of the model.

## 1.3 The coordinated model

To address the problems of the project ecosystem model, we propose a new model involving a central “coordinator” that a) provides a unified volunteer interface, and b) allocates computing power among a set of “vetted” projects. In the coordinated model the scientist and volunteer processes are as follows:

* Scientists can apply to the coordinator to have prospective projects pre-vetted. At that point they can be offered a certain amount of computing throughput. They can then proceed to create a BOINC project, without any risk. They don’t have to publicize their project, or even create a web site for it.
* In addition to single-group projects, we are enabling and encouraging the addition of VC back ends to existing HPC providers such as supercomputing centers and science gateways. The large population of scientists using these providers will benefit from VC (via shorter latencies and lower costs) with no effort on their part, and potentially with no knowledge that VC is being used.
* Volunteers interact through the coordinator web site. As part of the registration process, they indicate their “science preferences” – which areas of science they do or do not want to support – and their preferences for the location of the research. They attach their devices to the coordinator, which uses the AM architecture. They do not explicitly choose projects; rather, the coordinator dynamically decides what projects to attach each device to. It does this in a way that attempts to honors both volunteer preferences and allocation targets.

This model has a number of possible advantages:

* The risk of creating a project, and the need to publicize it, are eliminated. This will hopefully lead to more new projects.
* The coordinator’s “brand” acts as a unified brand for VC. Publicity campaigns (mass media, social media, co-promotions, etc.) can refer to this brand, rather than the brands of individual projects. This allows more effective promotion.
* It gives volunteers an interface defined in terms of science goals, which have been shown to be the most powerful incentive for participation. Also, compared to project-browsing, the interface is simpler.

# 2. Keywords

As a basis for volunteer preferences, we have defined a system of “keywords” for describing jobs. The system has the following structure:

* Category: currently “science area” or “location”. We could also consider categories for openness of results, but this may be hard to define.
* Hierarchy: each level N+1 keyword is a child of a single level N keyword. The hierarchy can change over time.
* Each keyword has a permanent integer ID, and short and long textual descriptions that can change over time.

## 2.1 Project and job attributes

Jobs can have an associated set of keywords. These are specified in the job submission process. For example, if a TACC job is submitted by a cancer researcher at UCB, the attributes would include “cancer research” and “UCB”.

Each project publishes a set of keywords, and for each keyword the estimated fraction of current jobs having that keyword. If the fraction is one, the keyword is implicitly associated with all the project’s jobs. The set of project keywords can change over time, reflecting changes in the project’s workload. SU polls projects for their keywords.

## 2.2 Keyword preferences

A volunteer can specify, on the SU web site, a set of “preferences”, represented as a map from keywords to {yes, no, maybe}. “No” means don’t send jobs with that keyword. “Yes” means preferentially send jobs with that keyword. A “no” for a keyword trumps “yes” for descendant keywords.

If a project has a keyword with job fraction 1, and a volunteer has “no” for that keyword, the volunteer’s devices should not be attached to that project.

When a new keyword is added, the default setting is “maybe” for all volunteers. Volunteers are notified of the new keyword in case they want to change this.

## 2.3 Keyword-based scheduling

The BOINC job dispatcher uses “score-based” scheduling: in deciding which jobs to send to a device, the scheduler computes a score for each job that includes a number of different factors; it then sends the highest-scoring jobs. We have extended this to include keywords. For each of the job’s keywords, if the volunteer has “yes” the score is incremented, and if “no” the score is zero (meaning don’t send the job).

This lets volunteer preferences be enforced for projects with applications in multiple science areas, and/or running jobs on behalf of a multiple client institutions.

# 3. Resource usage accounting

SU does accounting of resource usage. This serves two main purposes:

* It provides basis for volunteer incentives: for example, graphs showing work done recently, work “milestones”, and so on.
* It provides a basis for resource allocation to projects, and gives an estimate of the system-wide throughput.

BOINC has a sophisticated credit system for estimating the FLOPs performed by a completed job. It is fairly “cheat-proof”: it is difficult to get credit for computation not actually performed. However, this is based in part on job replication, meaning that credit for a job may not be granted until the companion job is completed, which could take weeks. This makes it undesirable for volunteer incentives.

Instead, SU uses a quantity called “estimated credit” (EC), which is maintained by the BOINC client on a per-job and per-project basis, based on the runtime of jobs and the peak FLOPS of the processors they use. EC is a cruder estimate than credit, and it is not cheat-proof. But it accumulates continuously, without waiting for job completion or validation.

# 4. The resource share model

SU’s resource allocation mechanism (see below) incorporates several factors. One of these is the notion of “resource share”: how much resources a projects gets relative to other projects. We use the following model:

* Each project P has a “resource share” RP. The fraction of resources available to P (over a sufficiently long period, and all other things, such as keywords, being equal) is at least RP/Rtotal, where Rtotal is the sum of RP over all projects. Resource shares may change over time.
* To enforce resource shares, each project P has a “balance” BP which represents how many FLOPs are “owed” to P. During a given accounting period (say, a day) BP is incremented by the number of FLOPs actually performed during the period, times RP/Rtotal . The balance is capped at a limit LP. Balance is in units of floating-point operation (FLOPs).
* When computation is reported by a client for a project P, BP is decremented by the amount of computation.
* At any point, computing resources are preferentially assigned to projects P for which BP is greatest.

This model handles both continuous and sporadic workloads well. For a project P with continuous workload, BP will remain around zero. For a project P with sporadic workload, BP will usually be at the limit LP. Then P generates a burst of work, it will have priority over the continuous-workload projects, and will get done quickly.

When a computer is assigned to a project, there will be a delay of about a day (the client polling period) until computation is reported by the client to SU. This means that the same project (the one for which BP is greatest) will be assigned to all hosts during that period. This is undesirable. To solve this problem, we maintain a separate “projected balance” for each project. This is decremented by the appropriate amount when a computer is assigned to the project. At the end of each accounting period, projected balances are reset to the balances.

## 4.1 Allocation

Initially, all projects will have equal shares (RP values). However, at some point we’ll want the capability of giving projects greater shares, on either a temporary or permanent basis. These “allocations” will be consist of elevating RP to a given, starting and ending at given times.

Allocation decisions will be made by the SU committee, according to a published policy. Allocations could be based on merit, on special requirements, or on payment.

## 4.2 Performance guarantees

Some existing HTC systems can offer “performance guarantees”: a client can be guaranteed a given throughput over a given period of time with very high probability. It’s desirable to offer analogous guarantees with VC resources.

The performance of a pool of volunteer computers can vary, in terms of both throughput and job latency. However, with a large pool, these quantities change slowly over time, and we can measure these changes and establish the statistics of their change. For example, given the total throughput T at a given time, we could find a T0 < T such that total throughput will remain above T0 for a week.

In this way, it’s likely that the “allocations” described above can be mapped to specific performance guarantees; the details remain to be figured out.

Such guarantees would be project-level. Can we provide performance guarantees to a particular job submitter within a project that serves lots of job submitters? This is more complex but it may be possible. The BOINC server software uses the linear-bounded model, exactly as described above, to allocate resources among computing job submitters within the project. It’s possible that the combination of a project-level allocation and a submitter-level allocation provide some form of performance guarantee to the submitter.

# 5. Resource allocation

The central function of SU is to allocate resources among projects. It does this by dynamically assigning projects to computers.

The resource allocation policy has several possibly conflicting goals:

* It must enforce volunteer preferences: if 100% of volunteers said “no” to cancer research, SU can’t assign computers to a project that does only cancer research. However, we anticipate that volunteer preferences will be primarily positive (“yes”). This gives SU some freedom, since “yes” is non-binding: SU is free to assign “maybe” work even if “yes” work is available.
* It tries to divide resources among projects based on the “resource share” model described above.
* It tries to maximize total throughput. For example, if a host has a GPU, it should be assigned at least one project that can use the GPU.

Clients using SU periodically (perhaps once per day) issue an AM RPC. The request message includes a list of currently-attached projects and their work totals; these are used to update account records. The reply message includes a list of projects to attach to. The client detaches from any projects not on this list. For each project, the reply specifies a “resource share”: a value of zero means that the client should do work for this project only if none of the other projects have work available.

Being attached to a project has a disk overhead; the client caches applications files for the project, which may include large VM image files. Hence we want to limit the number of projects each client is attached to. On the other hand, if a project has a large disk footprint on a client, we may want the client to remain attached (with a zero resource share) even if we don’t want it to compute at this point.

## 6. The project assignment algorithm

With these factors in mind, here is a sketch of the project assignment algorithm currently used by SU:

* For each project (of all the projects managed by SU) compute a score for this client. This score includes several components:
  + Don’t use the project if it doesn’t support the host’s platform.
  + A keyword factor. Increment the score if the project has keywords in the volunteer’s “yes” list. Don’t use the project if it has a “no” keyword with job fraction one.
  + Increment the score if the host has VirtualBox and the project can use it.
  + Increment the score if the host has a GPU that the project can use.
  + Add the project’s allocation balance.
  + Increment the score if the host is already attached to the project.
* Choose the 3 highest-scoring projects. If the host has a GPU that none of these can use, add the highest-scoring project that can use the GPU.
* If the user doesn’t have an account on a selected project, initiate an account creation and tell the client to retry in 1 minute.
* If the client is currently attached to a project not in the top 3 but whose disk footprint exceeds 10 MB and whose score is nonzero, tell the client to remain attached with zero resource share.

The algorithm has lots of undetermined parameters. We’ll guess appropriate values. An interesting research project would be to create a simulator for studying system behavior with different algorithms and parameters.

# 8. Implementation notes

We are using existing BOINC web code (PHP, database schema) as a basis for implementing SU. The BOINC web code (used for project web sites) has many features needed by SU:

* Account creation and login.
* Computing preferences.
* List of hosts.
* Message boards, private messages, “friends”, and other community-oriented features.

The code is configurable so that features related to job processing (which are not relevant to SU) can be disabled.

We have added a number of web pages and tables that are specific to SU: projects, allocations, project and volunteer keywords, project accounts, accounting records, and so on.

We have implemented a separate web interface for SU admins, primarily for viewing graphs and tables of accounting data.

# 3. Science United

I am currently implementing a coordinator for volunteer computing, called Science United (<https://scienceunited.org>). This project is funded by the National Science Foundation. Science United is operational and is being tested prior to a public launch.

# 2 The coordinator committee

We will establish a “coordinator committee” to determine coordinator policies, including project vetting and resource allocation. The committee may include representatives of the U.S. and European scientific funding agencies, leaders of the coordinator project, and members of the volunteer community.

## 2.1 Project vetting

The committee must decide what projects to vet. This will be based on published criteria such as:

* The project’s computing is toward a scientific or technical goal (broadly interpreted to include things like mathematics and cryptography).
* The project is non-commercial.
* The project’s leadership has a certain level of qualification (as demonstrated, e.g., by publications).
* The project can prove that it understands and follows certain security practices such as code-signing.

The committee will define a process by which potential new projects can apply for vetting. A scientist could apply for vetting, then submit a grant proposal to fund the development of the project.

The committee may choose to charge fees for vetting and/or allocation, with the possibility of waiving the fee in special cases. The proceeds would go toward the development and maintenance of the coordinator software, and to server costs.