

Balancing Scientist Needs and Volunteer Preferences in Volunteer Computing Using Constraint Optimization

James Atlas, Trilce Estrada, Keith Decker, Michela Taufer
University of Delaware, U.S.A.
{atlas, estrada, decker, taufer}@cis.udel.edu

Abstract. BOINC is a middleware for Volunteer Computing. In BOINC projects, heterogeneous resources distributed across the Internet are used for large-scale scientific simulations. The large need for resources in BOINC projects often competes with volunteer preferences: volunteers can impose limits on the use of their idle resources. Most of the time, maximum project performance can be achieved only when volunteer preferences are neglected.

To address this problem, we propose a novel optimization procedure based on constraint optimization techniques that actively allocates volunteer resources to improve project throughput and, at the same time, aims to preserve volunteer preferences. We evaluate our approach against the current allocation strategies of BOINC using EmBOINC, a full-scale emulator of BOINC projects using realistic trace populations of volunteer clients (including heterogeneity, availability, reliability). We show the increase in project throughput obtained with our approach and discuss the trade-off between volunteer preferences and project throughput.

1 Introduction

Volunteer Computing (VC) is a form of distributed computing in which ordinary people (i.e., volunteers) volunteer processing and storage resources to computing projects. BOINC is a well-known middleware for VC [1] supporting scientific computing projects (e.g., physics, biology, and medicine). The main strength of BOINC systems is its capability to provide scientists with PetaFLOPs of computing power at low cost. The VC community powered by BOINC currently counts approximately 50 projects and 580 000 volunteer computers supplying an average of 1.2 PetaFLOPs to these projects.

VC resources increasingly include diverse platforms such as video game consoles (Playstations) and graphics processing units (GPUs). Some VC projects are able to customize their code to benefit from performance features of these platforms. This creates an instance of a general resource allocation problem where jobs have disparate performance profiles depending on the platform of execution. In addition, volunteers can specify resource allocation preferences over a subset of VC projects that they want to participate in, essentially constraining the possible jobs the server can allocate to a volunteer host and ultimately compromising the throughput of projects. In other words, maximum project performance, measured in terms of project throughput, is hindered by volunteer's preferences. On the other hand, ignoring volunteer's preferences for performance sake can upset the donors who can ultimately withdraw their resources. Server scheduling policies decide which jobs to assign to volunteer hosts given a set of unallocated jobs and volunteer preferences. Ideally these policies should optimize allocation

of jobs across heterogeneous volunteer resources and, at the same time, preserve the volunteer’s preferences in the best way and with the highest performance. This task is made more and more challenging by the increasing heterogeneity of VC systems.

To address this challenge, we propose a novel optimization procedure that actively allocates volunteer resources to improve project throughput and preserve volunteer preferences. Our optimization procedure is based on constraint optimization techniques (COP) and provides a robust framework for maximizing the contributions of largely diverse heterogeneous resources. We evaluate our approach against the current, most advanced allocation strategies of BOINC using EmBOINC, a full-scale emulation of the BOINC platform using realistic trace populations of volunteer’s hosts (including resource heterogeneity, churn, availability, reliability). This paper shows the increase in project throughput obtained with our approach and discusses the trade-off between volunteer preferences and project throughput.

This paper is organized as follows: Section 2 presents a short overview of important background concepts such as VC, BOINC, our emulation of BOINC projects, and COP. In Section 3 we introduce our optimization procedure. Section 4 compares our approach with the current practice scheduling policies of BOINC. Section 5 concludes the paper and presents some future work.

2 Background and Related Work

2.1 Volunteer Computing

Volunteer Computing (VC) projects employ computing resources (e.g., desktops, notebooks, and servers) owned by ordinary people and connected to the Internet. Traditionally, VC projects target large search problems in science and, therefore, generate large sets of jobs that are distributed across VC resources. Replication of jobs is used to address the volatility of these systems as well as other issues like malicious attacks, hardware malfunctions, or software modifications that ultimately affect the reliability of results. Replicas of jobs (also called job instances) are distributed to different VC resources (also called hosts) that execute them. When finished, the hosts send their results to the project server, which collects the results and distinguishes between successful and unsuccessful results. Unsuccessful results are those that either are erroneous or are returned too late, i.e., timed-out.

2.2 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [1] is an open-source system that harnesses the computing power and storage capacity of thousands or millions of PCs owned by volunteers for scientific simulations. The computing resources available to a BOINC project are highly diverse: the hosts differ by orders of magnitude in their processor speed, available RAM, disk space, and network connection speed. Some hosts connect to the Internet by modem only every few days, while others are permanently connected. Recently, the heterogeneity of BOINC platforms has been enriched by the adding of GPUs and Playstations. The BOINC model involves projects

and volunteers. Projects are organizations (typically academic research groups) that need computing power. Projects are independent (each operates its own BOINC server) and have different resource requirements. Volunteers participate by running the BOINC client software on their computers (hosts). Volunteers can attach their hosts to one or multiple projects (preferred projects). When a BOINC client is attached to a project, it periodically issues a scheduler *request* to the project's server. The request message includes a description of the host and its current workload (jobs queued and in progress), descriptions of newly-completed jobs, and a request for new jobs based on the volunteer's preferences. The *reply* message from the server may contain a set of new jobs. Multiple job results may be returned; this reduces the rate of scheduler requests and accommodates clients that are disconnected from the Internet for long periods.

2.3 Scheduling in BOINC

Initially BOINC scheduling policies relied on greed and naive policies. Recently, more sophisticated server-side scheduling policies have been implemented in several BOINC projects. Currently, World Community Grid has a number of criteria for job assignment [2], based on host and job diversity (e.g., size of the job and speed of the host relative to an estimated statistical distribution, disk and memory requirements for the job to be completed, homogeneous redundancy [3] and host error rate). A scoring-based scheduling policy uses a linear combination of these terms to select the best set of jobs that can be assigned to a given host. Projects can adjust the weights of these terms, or they can replace the scoring function entirely. None of these policies search for trade-off between volunteer preferences and project requirements. Trade-offs are less likely when the level of heterogeneity of the resource population and their diversity increase (e.g., when GPUs are also part of the available hosts).

2.4 Emulating BOINC

The scheduling policies embedded in the BOINC server have a large impact on the project throughput and other performance metrics. Unfortunately, it is difficult (if not impossible) to do controlled performance experiments in the context of a large VC project because there are many factors that cannot be controlled and because poorly-performing mechanisms can waste lots of resources driving away volunteers. On the other hand, exploring new policies and tuning setting parameters for high performance can be done in simulated environments, where it is possible to test a wider range of hypotheses in a shorter period of time without affecting the BOINC community. To evaluate our scheduling approach we use EmBOINC (Emulator of BOINC Projects), a trace-driven emulator that models heterogeneous hosts and their interaction with a real BOINC server [4]. By plugging into a BOINC server, EmBOINC triggers the server's daemons to generate and distribute jobs to the EmBOINC hosts. EmBOINC uses statistical information obtained from real BOINC traces to characterize volatile, heterogeneous, and error-prone hosts. As it occurs in real BOINC projects, EmBOINC can emulate different projects running simultaneously. Projects can share the simulated hosts partially or completely. For every project, the associated hosts can have different levels of heterogeneity, errors, availability, and reliability. Using EmBOINC, different patterns

for job generations as well as different policies for job distribution and validation can be studied.

2.5 Constraint Optimization

Many historical problems in the AI community can be transformed into Constraint Satisfaction Problems (CSP). Early domains for constraint satisfaction problems (DisCSP) included job shop scheduling [5] and resource allocation [6]. Many domains for distributed systems, especially teamwork coordination, distributed scheduling, and sensor networks, involve overly constrained problems that are difficult or impossible to satisfy for every constraint. Recent approaches to solving problems in these domains rely on optimization techniques that map constraints into multi-valued utility functions. Instead of finding an assignment that satisfies all constraints, these approaches find an assignment that produces a high level of global utility. This extension to the original CSP approach has become popular in distributed systems, and has been labeled the Distributed Constraint Optimization Problem (DCOP) [5]. A typical constraint optimization problem (COP) begins with a constraint graph mapping of a problem. The COP mapping is defined as a set of n variables and m constraints producing the tuple $\langle X, D, U \rangle$ where:

- $X = \{x_1, \dots, x_n\}$ is a set of variables, each one assigned to a unique agent
- $D = \{d_1, \dots, d_n\}$ is a set of finite domains for each variable
- $U = \{u_1, \dots, u_m\}$ is a set of utility functions such that each function involves a subset of variables in X and defines a utility for each combination of values among these variables

An optimal solution to a COP instance consists of an assignment of values in D to X such that the sum of utilities in U is maximal. An example COP instance for the standard graph coloring problem with weighted utilities is shown in Figure 1.

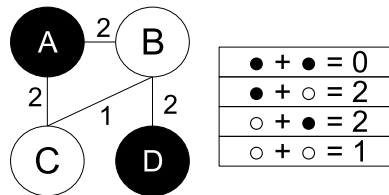


Fig. 1. COP Example: Simple graph coloring problem with utility functions. Coloring shown is optimal for this problem, and utility values are placed alongside the constraints.

3 Methodology

The idea behind our approach is that we can increase throughput for BOINC projects by intelligently coordinating schedules for volunteers. To achieve this, we develop a

constraint optimization (COP) mapping that pursues high throughput while trying to adhere to the volunteer’s preferences.

3.1 Approach Overview

In Figure 2 we show the different components of our solution to optimize BOINC schedules. The process can be run continuously in a real-time environment because it only interacts with the BOINC server through its database. Thus, the flow through the diagram can be considered a cycle, beginning with the input factors and ending with an update to the BOINC database. The input factors are passed to the mapping layer (mapping application) to convert the actual BOINC scheduling data and parameters into a constraint graph. The constraint graph is used as a general representation for a COP and is passed into the optimization algorithm. The optimization algorithm determines a new containing allocation schedules for each volunteer’s resources to the BOINC projects. These allocation schedules are then updated in the database and are used to determine which jobs to return when a volunteer requests new work. We will describe in detail each of these steps in the following sections.

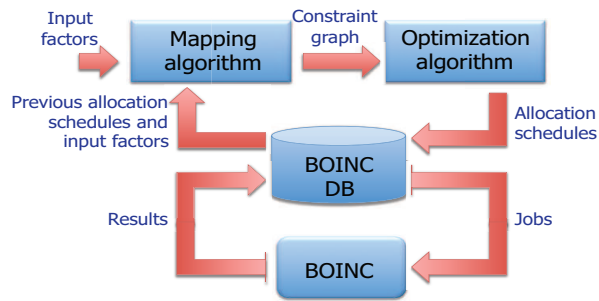


Fig. 2. Overview of our constraint optimization approach

3.2 Input Factors

The first step in our approach is to define input factors that we will consider for the optimization. The value of each factor directly affects the outcome of the optimization procedure. A list of these factors and the actor responsible for providing their value appears in Table 1. In addition to these definable factors, input values are also derived from the BOINC database about volunteer resources (e.g., estimated flops) and project characteristics (e.g., GPU/coprocessor support).

3.3 Mapping BOINC Schedules to COP

The next step is to map the input factors into a coherent problem representation. We use a constraint optimization representation, so we will provide a mapping from the

ID	Factor	Actor	Description
VP	Volunteer Preferences	Volunteer	Set of projects preferences for a volunteer. Volunteers assign 1 to preferred projects and 0 otherwise.
PTN	Project Throughput Need	Scientist	A target number of results returned per project per day. Specified in GFLOPS/s (or using deadlines on a set of jobs).
AS	Allocation Schedule	BOINC-COP	The specific allocation of volunteer resources to different projects. The current set of allocation schedules is input and a new set of allocation schedules is output. Percent per project per volunteer.
TvP	Throughput vs. Preference	Scientist	A weight between 0 and 1 for favoring project throughput and for volunteer preferences, with total weight adding to 1, where 0 means full matching of volunteer preferences and 1 complete ignore.

Table 1. Input factors

input factors to a constraint graph. A constraint graph contains variables as nodes and constraints as edges. We consider two types of variable nodes:

Allocation Schedule (AS) represents the allocated volunteer schedule. The AS factor listed earlier determines the starting value for this variable. Possible assignments for this variable represent new allocation schedules for the volunteer.

Project Throughput Need (PTN) represents a level between 0 and 1 to which the project needs additional volunteer resources. In relation to the TN factor listed earlier, a value of 1 means that the project needs additional resources and is currently short of its target. A value of 0 means that the project has no use for additional resources and has already met its throughput target. A value in between means that a project has met its throughput target but could use additional resources.

Each volunteer has one AS variable and each project has one PTN variable. We now create binary constraints between each volunteer (AS variable) and every project (PTN variable) the volunteer is willing to work for. This constraint returns a utility value that represents the utility of a volunteer's current allocation schedule for a given project's level of throughput need. The value of this constraint, $U(N, M)$ for project N and volunteer M is:

$$U(N, M) = P_N(AS_M) \cdot PTN_N \cdot C_M(N) \cdot (W_{AT} + (1 - W_{AT}) \cdot VP_M(N)) \quad (1)$$

Where:

- $P_N(AS_M)$ is the percent allocated to project N in the schedule of M
- PTN_N is the level of throughput need for project N (between 0 and 1)

- $C_M(N)$ is the contribution of volunteer M to project N in GFLOP/s
- W_{AT} is the weight given to optimizing project throughput (between 0 and 1)
- $V P_M(N)$ is the volunteer preference of volunteer M given to project N as a score (between 0 and 1)

These variables and constraints form the constraint graph representation of our original input factors. We now take this general COP representation and apply our optimization algorithm to find a high utility scheduling policy.

3.4 Optimization Algorithm

Our optimization algorithm takes as input the constraint graph formed in the previous section, and solves for a new scheduling policy containing allocations that optimize each volunteer's resources to the BOINC projects. Our algorithm is a modified version of the stochastic gain algorithm described in [7]. We add support for derived variables and random gain delays. The following steps are performed in parallel for each variable:

1. Get max local gain (best volunteer schedule for current state of PTN variables)
2. With probability p , change to max assignment (new value for AS variable; setting p too high can prevent full convergence)
3. Derive new values from neighbors (for PTN variables from all volunteer schedules)

These three local search steps are performed for a number of cycles; at a specified maximum amount of time the algorithm is terminated and the best utility assignment encountered so far is chosen. The algorithm converges with low enough p . We implemented a delay in number of cycles between changes to the same AS variable to also help with convergence. The algorithm can scale to tens of thousands of variables. If we require optimization of larger sets of volunteers, we can pre-process the set and cluster volunteers into groups that share similar preferences and resource contribution characteristics. Then we simply treat each group as a super-volunteer with one volunteer variable which represents an identical allocation schedule assigned to all volunteers in the group.

3.5 Integration with BOINC Server

The output from the optimization algorithm is a set of allocation schedules for each volunteer. We store these allocation schedules in the BOINC database. Modifying the BOINC server to use our allocation schedules is easily done. The BOINC scheduler uses a scoring mechanism to handle each request for work from a volunteer. Each unassigned job receives a score for possible assignment to the requesting volunteer. The highest scoring set of jobs that fill the amount of time requested by the volunteer are sent. To integrate our allocation schedules we simply add a value to the score if the job matches the volunteer's allocation schedule. The schedule contains a number between 0 and 1 for each project for this volunteer. We generate a random number between 0 and 1 and if it is less than the schedule allocation number than that job receives a higher score. Thus, over time the allocation of jobs will match the percentages specified in the schedule. Typically the optimization process, run in parallel with the BOINC scheduler, took less than one minute to complete. Large real-time systems can tune the frequency of optimization, or to produce approximate optimizations in a shorter amount of time.

4 Evaluation

To evaluate our approach, we use EmBOINC and consider different scenarios matching the behavior of real BOINC projects.

4.1 Performance Metrics

With EmBOINC it is possible to conduct extensive experiments targeting different performance metrics. We measure the following metrics:

Throughput-based metrics include *project throughput* in terms of total results returned to the scientist or results returned per day.

Preference metrics count the total number of jobs executed by volunteers for preferred and non-preferred projects.

Deadline-based metrics count the number of job results that completed prior to the deadline given by the scientist.

4.2 Scenarios

We tested our approach in two orthogonal scenarios with respect to the way jobs are generated. We considered three projects running simultaneously.

Scenario 1 In this scenario we tested the ability of our solution to optimize throughput with *uniform generation of jobs*. Project 1 (uses only CPU) and 3 (uses CPU or GPU) were given a target of 20% of the overall throughput each. Project 2 (uses only CPU) targeted 60% of the throughput. The scenario generated 4000 jobs for Projects 1 and 3 and 12000 for Project 2. This scenario is typical for projects without deadlines like SETI@Home (<http://setiathome.berkeley.edu>).

Scenario 2 In this scenario we tested the ability of our solution to optimize throughput with *irregular generation of jobs*. For Project 2 we randomly injected 8 batches of 1500 jobs over 25 days, with a 10 day deadline for each batch. For Projects 1 and 3 we kept a uniform generation of 4000 jobs each. This scenario is similar to the real-world case in Critical Assessment of techniques for protein Structure Prediction (CASP). During the biennial CASP competition (<http://predictioncenter.org/>), new targets (amino acid sequences) are released to the participants almost every day with a deadline of 15 days for the target 3D prediction. Projects such as Predictor@Home and Rosetta@Home belong to this class of scenarios.

We ran our simulations using a fixed amount of simulated time (25 days). We used a base set of 500 volunteers, of which 20% have GPUs. Each volunteer randomly chose 1 or 2 preferred projects of the 3 possible. All other host characteristics (e.g., CPU speed, memory) were randomly generated based on trace data from real-world BOINC projects.

4.3 Results

Figure 3(a) shows total throughput (number of results) for the three projects in Scenario 1 using BOINC and different levels of TvP for our constraint optimization (COP) approach. A low TvP means closer matching to volunteer preferences, where TvP equal to zero means perfect matching. The gray bar is jobs executed by volunteers for their preferred projects, and the white bar is jobs executed for non-preferred projects. If we fully comply with volunteer preferences (TvP=0), our system has the same performance as BOINC. As we increase the TvP factor, we increase the total throughput of the system, but we diverge from volunteer preferences. A complete violation of volunteer preferences is not needed to achieve high levels of throughput, as TvP setting of 0.25 already achieves higher throughput than BOINC (+12%). Figure 3(b) shows the same metric for

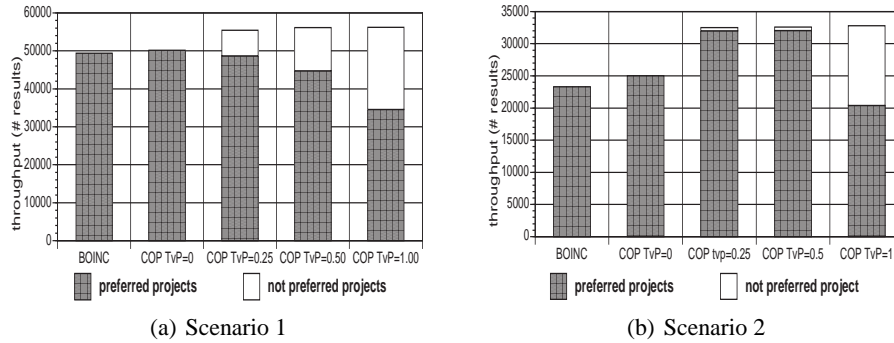


Fig. 3. Throughput (number of results) for the two different scenarios

the project with deadlines in Scenario 2. In this scenario, batches of jobs are randomly injected, with a 10 day deadline for each batch. Throughput is the number of results for Project 2 returned to the BOINC server before their deadline. As in Scenario 1, the gray bar is jobs executed by volunteers for their preferred projects, and the white bar is jobs executed for non-preferred projects. If we fully comply with volunteer preferences (TvP=0), our system performs slightly better than BOINC (+7.4%). As we increase the TvP factor, the throughput increases significantly. Again, we do not need to completely violate the volunteer preferences; with a TvP setting of 0.25 we gain 39.6% throughput. Increasing the TvP factor allows our system to re-allocate volunteer resources to jobs with upcoming deadlines. Note that the percentage of jobs performed for non-preferred projects is minimal (1.6%).

Overall, we see that our approach increases throughput for both uniform and irregular job generation scenarios. Increasing the TvP factor allows us to achieve higher throughput at a cost of making volunteers execute jobs for non-preferred projects. A TvP trade-off of 0.25 provides maximum throughput with minimum volunteer preference violation. The reason for increase in total throughput is *not* due to additional idle cycles (we had the same amount of resource idle time for the several runs); it is because we use the resources more efficiently. For both BOINC and our approach, we made

sure that there were new jobs available for every volunteer request. The results were validated by repeating each simulation three times; each time we observed the same behavior.

5 Conclusions and Future Work

We presented a novel optimization procedure based on constraint optimization techniques that actively allocates volunteer resources to improve project throughput and, at the same time, aims to preserve volunteer preferences. We show two scenarios that exhibit increased project throughput (up to 39.6%) for a minimal trade-off in execution of jobs for non-preferred projects. Our results show that it is possible to balance the needs of scientists with the preferences of volunteers in VC projects. In the future, we intend to extend our approach to include optimization of an expanded definition of volunteer credit. Currently, volunteer credit is based on the number of FLOPS executed by the volunteer for an project. However, in some scenarios scientists may want to assign greater credit per FLOP for one project than another. In this case the volunteer would want to optimize the amount of credit they receive for their resource contribution. This is similar to some scenarios in grid and cloud computing scheduling, and we intend to examine how our approach can be applied to these related problems.

Acknowledgment

This work was supported by the National Science Foundation, grant #OCI-080265, DAPLDS - a Dynamically Adaptive Protein-Ligand Docking System based on multi-scale modeling.

References

1. Anderson, D.P.: BOINC: A System for Public-Resource Computing and Storage. In: Proc. of the 5th IEEE/ACM International Workshop on Grid Computing. (2004)
2. Anderson, D.P., Reed, K.: Celebrating Diversity in Volunteer Computing. In: Proc. of the Hawaii International Conference on System Sciences (HICSS). (2009)
3. Taufer, M., Anderson, D., Cicotti, P., C.L. Brooks III: Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results Using Public Computing. In: Proc. of the 14th Heterogeneous Computing Workshop. (2005)
4. Estrada, T., Reed, K., Anderson, D., Taufer, M.: Emboinc: An emulator for performance analysis of boinc projects. In: Proc. of PCGrid. (May 2009)
5. Liu, J., Sycara, K.P.: Exploiting problem structure for distributed constraint optimization. In: ICMAS 95. (1995) 246–254
6. Modi, P.J., Jung, H., Tambe, M., Shen, W.M., Kulkarni, S.: A dynamic distributed constraint satisfaction approach to resource allocation. In Walsh, T., ed.: CP. Volume 2239 of Lecture Notes in Computer Science., Springer (2001) 685–700
7. Zhang, W., Xing, Z., Wang, G., Wittenburg, L.: An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In: AAMAS 03. (2003)