J Grid Computing
DOI 10.1007/s10723-009-9126-3

Performance Prediction and Analysis of BOINC Projects: An Empirical Study with EmBOINC

Trilce Estrada · Michela Taufer · David P. Anderson

Received: 21 February 2009 / Accepted: 4 August 2009

Abstract Middleware systems for volunteer computing convert a set of computers that is large and diverse (in terms of hardware, software, availability, reliability, and trustworthiness) into a unified computing resource. This involves a number of scheduling policies and parameters, which have a large impact on the throughput and other performance metrics. How can we study and refine these policies? Experimentation in the context of a working project is problematic, and it is difficult to accurately model complex middleware in a conventional simulator. Instead, we use an approach in which the policies being studied are "emulated", using parts of the actual middleware. In this paper we describe EmBOINC, an emulator based on the BOINC middleware system. EmBOINC simulates a population of volunteered clients (including heterogeneity, churn, availability, and reliability) and emulates the BOINC server components. After describing the design of EmBOINC and its validation, we present three case studies in which the impact of different scheduling policies are quantified in terms of throughput, latency, and starvation metrics.

Keywords Volunteer Computing · Docking@Home · World Community Grid · Simulation · Emulation

This material is based upon work supported by the National Science Foundation, grant #OCI-0802650, DAPLDS—a Dynamically Adaptive Protein-Ligand Docking System based on multi-scale modeling and grant #OCI-0721124 Middleware for Volunteer Computing, and by the CONACyT fellowship #171595.

T. Estrada · M. Taufer (✉)
University of Delaware, Newark, DE, USA
e-mail: taufer@udel.edu

T. Estrada
e-mail: estrada@udel.edu

D. P. Anderson
U.C. Berkeley Space Sciences Laboratory, Berkeley, CA, USA
e-mail: davea@ssl.berkeley.edu

1 Introduction

Volunteer Computing (VC) is a form of distributed computing in which computer owners volunteer resources to scientific computing projects. BOINC is a software platform for volunteer computing [1] that is used by projects in physics, molecular biology, medicine, chemistry, astronomy, climate study, mathematics, and other fields. BOINC-based projects derive their power from heterogeneous computing resources with varying levels of availability and reliability [8]. The projects vary in terms of job length, sensitivity to errors, and requirements for result verification (such as the number of replicas that are

Published online: 19 August 2009

compared for agreement before being accepted). In the past five years, the VC community based on BOINC has grown significantly and currently there are approximately 50 projects and 580,000 volunteer computers supplying an average of 2.1 PetaFLOPS.

BOINC consists of separate client and server components, each of which embodies a set of scheduling policies. The client policies have been studied elsewhere [3]; here we are concerned with the server policies. The server component of BOINC embodies a number of scheduling policies and parameters that 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 volunteer computing project: there are many factors that cannot be controlled, and poorly-performing mechanisms can waste significant amount of resources (and potentially drive away volunteers). Simulation environments, on the other hand, make it possible to explore new policies, optimize parameters, and test a wide range of hypotheses in a short period of time without affecting the volunteer base. A simulation environment can be based on a "pure simulator" implementing an abstract model of a system, an "emulator" using all or part of a real system, or a hybrid approach combining the two. For a large, complex system like BOINC, pure simulation is infeasible: it would be difficult to accurately represent the behavior of a BOINC server in an abstract model, and to track the frequent modifications to the BOINC software. A better approach is to use emulation for the portion of the system under study, and simulation for the rest of the system.

In this paper we present EmBOINC, a trace-driven emulator of BOINC projects. EmBOINC is intended to allow developers to tune existing server scheduling policies and to accurately predict the performance of new policies. EmBOINC uses emulation for the server, and it uses simulation to represent the heterogeneous, volatile population of volunteers' hosts. EmBOINC interacts directly with the actual BOINC server daemons, triggering the generation, distribution, collection, and validation of jobs. By using emulation in this way, EmBOINC minimizes the maintenance burden caused by the rapid development of BOINC. At the same time, by plugging directly into a BOINC server, EmBOINC allows testers to directly and accurately tune BOINC policies for different host populations. After describing the design of EmBOINC and its validation, we present three case studies in which the impact of different scheduling and validation policies are quantified in terms of throughput, latency, and utilization metrics. This expands upon our previously reported work [10], in which we presented preliminary validation results of EmBOINC.

This paper is organized as follows: Section 2 presents a short overview of VC and BOINC. The EmBOINC framework and its software components are described in Section 3. In Section 4 we validate EmBOINC by comparing its performance predictions with those of a real BOINC project. Section 5 presents three case studies in which the impact of different scheduling policies, homogeneous redundancy granularity, and job replication policies are empirically studied. Section 6 compares EmBOINC with other work. Section 7 concludes the paper.

2 Volunteer Computing and BOINC

2.1 Volunteer Computing

Volunteer Computing (VC) projects employ "hosts" (for example desktops, notebooks, and servers) owned by the general public and connected to the Internet. The computing resources 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. VC projects typically run applications with large numbers (as many as millions per day) of independent, compute-intensive jobs.

2.2 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [1] is an open-source system for volunteer computing. The BOINC model involves "projects" and "volunteers". Projects are organi-

zations (typically academic research groups) that need computing power. Projects are independent; each operates its own BOINC server. Volunteers participate by running the BOINC client software on their computers (hosts). Volunteers can "attach" each host to any set of projects, and can specify the quota of bottleneck resources allocated to each project.

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. The *reply* message may contain a set of new jobs. Multiple jobs may be returned; this reduces the rate of scheduler requests and accommodates clients that are disconnected from the Internet for long periods. BOINC provides the option of using job replication to address issues like malicious attacks, hardware malfunctions, and software modifications that affect the reliability of results. Replicas of jobs (also called job instances) are distributed to hosts. When finished, the hosts send their results to the project server. The project server classifies results as valid or invalid based on their agreement with other instances.

A BOINC server is centered around a relational database, whose tables correspond to the abstractions of the BOINC's computing model, such as platforms, applications, application versions, jobs, and job instances. BOINC daemons are in charge of generating new jobs (work generator); periodically replenishing a shared memory cache with job instances from the database (feeder); changing the status of jobs and job instances in the database (transitioner); validating results (validator); handling validated results (assimilator) and removing the files associated with completed jobs (file-deleter).

The BOINC server software embodies many sophisticated scheduling policies. For example, there are a number of criteria for job assignment [2], based on host and job diversity (for example 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 [23] 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. The benefits of these terms and the different scoring functions can be quantified with an emulator such as EmBOINC.

2.3 BOINC Projects

In this paper we consider two BOINC projects and their traces collected over time. The projects are: the IBM World Community Grid[1] and the Docking@Home[2] project.

World Community Grid (WCG) is an initiative supported by IBM that makes VC computing available to not-for-profit organizations. WCG currently supports several simultaneous applications. In our work we consider two of these applications, FightAIDS@Home and Human Proteome Folding Phase II. FightAIDS@Home searches for drugs to disable HIV-1 Protease. The two major objectives of Human Proteome Folding Phase II are: to obtain higher resolution structures for specific human proteins and pathogen proteins, and to further explore the limits of protein structure prediction.

Docking@Home (D@H) is a BOINC project at the University of Delaware. By searching the large space of potential ligand conformations, D@H reduces the time and cost needed to design new drugs by several orders of magnitude. In this search, the protein and ligand can have different size and flexibility and their characteristics can have large impact on job duration.

3 Methodology

To analyze and predict performance of BOINC projects, we implemented and used EmBOINC (Emulator of BOINC Projects) [10]. EmBOINC is a trace-driven, hybrid emulator that models heterogeneous hosts and their interaction with a BOINC server. EmBOINC combines the accuracy of using

[1] http://worldcommunitygrid.org

[2] http://docking.cis.udel.edu

part of an existing system (in this case the BOINC server) with the efficiency of simulating a very large number of hosts interacting with the server.

BOINC clients communicate with servers using HTTP. In particular, they periodically issue a Web RPC to the projects scheduler. In this RPC, the request message contains a list of completed jobs and a request for more jobs; the reply message contains a list of new jobs. We used this RPC mechanism as the interface between the two parts of EmBOINC: the client simulator sends a series of RPCs to the server emulator. For efficiency, we execute the scheduler locally rather than create a connection for each request.

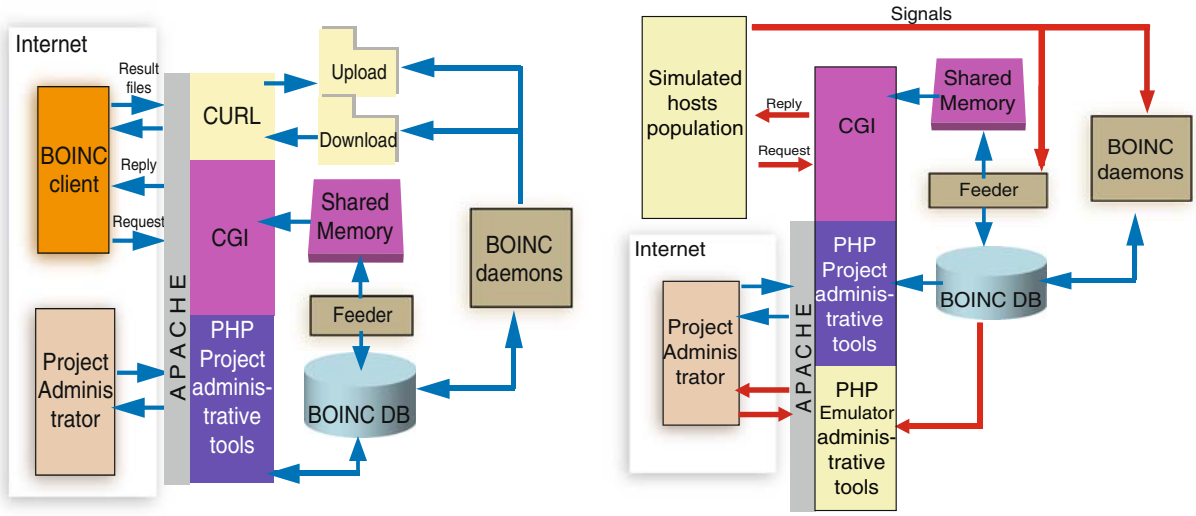
3.1 Emulator Framework

A BOINC server consists of a relational database and several programs that communicate through the database and shared memory. One of these programs is the scheduler, which is run by the web server as a CGI program. The others are daemons, which perform tasks such as job generation, validation, retry of timed-out jobs, cleanup of files and database, and so on. These daemons are driven by the passage of time; for example, the retry generator wakes up every 5 seconds and checks for jobs that should have been completed by the current time. The server part of EmBOINC uses the same set of processes as a real BOINC server, including the database server. To enable faster-than-real-time simulation, we made two changes to the server code. First, we changed the scheduler so that it can handle a sequence of RPC requests as described above. Second, we added a virtual time mechanism to the BOINC daemons; instead of sleeping for a time interval, they wait for a signal, then read the current simulated time from a file. Both of these changes are in the BOINC source code, conditionally compiled. Thus EmBOINC can always be used to study the latest version of BOINC.

Figure 1 shows the server and client interaction in a BOINC system (Fig. 1a) and the changes to the system in EmBOINC (Fig. 1b). First, we replace the BOINC clients with a simulated population of hosts (see Section 3.2). Second, we introduce signals to control the daemons. Finally, we extend the BOINC interface to access information on the simulated project.

3.2 Simulation Component

EmBOINC uses simulation to model the population of volunteer hosts. The behavior of these hosts is managed by a discrete event simula-



(a) BOINC server and client

(b) BOINC server + EmBOINC

Fig. 1 BOINC platform with and without EmBOINC (**a**, **b**)

tor (DES). Discrete event simulators consist of events and entities [11, 13, 20, 21]. Entities trigger events [13]. EmBOINC entities are: *Init Entity*, *EmBOINC DES Controller*, *Host Generator*, *Simulated Host*, *Job Generator*, and *Simulated User*. Simulated hosts and simulated users are also referred to as hosts and users respectively. Entities are characterized by attributes and are stored in a priority queue. The priority queue orders entities based on their next activation time. Each entity is associated with one or more events. Events are actions triggered by the associated entity at a certain point in time that cause changes in the simulated state and modifies the priority queue.

A list of events associated with EmBOINC entities is shown in Table 1. In the table, entities are in bold; an event is denoted by a number; a group of events is enclosed in parenthesis; a group of events that may or may not occur depending on the entity state is enclosed by brackets; a star denotes an event or group of events occurring zero or more times; and a plus sign denotes an event or group of events occurring one or more times. The entity *Simulated User* is a passive entity and does not trigger any event therefore is excluded from the table.

Table 1 Table of events triggered by entities

EmBOINC Init	**Job Generator**
1 Set next time	5 Read traces
2 Get entity's time	6 Read job
3 Enqueue	7 Make job
4 Terminate	
EmBOINC DES controller	**Host generator**
5 Read configuration	5 Read traces
6 Read traces of users	6 Read host
7 Read user	7 Make host
8 Make user	
9 Update simulated time	**Simulated host**
10 Pop entity from queue	5 Write request
11 Execute entity	6 Send request
12 Call transitioner	7 Parse reply
13 Call feeder	8 Get job
14 Call validator	9 Get job instance
15 Call assimilator	10 Execute job instance
16 Wait for signal	11 List current job load
17 Continue simulation	12 Calculate requested job
Sequence of events	
EmBOINC DES controller:	
$5, 6, (7, 8)^{+}, 9, ((10, 9, 11, [12, 13])^{+}, 12, 13, 14, 15, 16, 17)^{+}, 4$	
Work generator: $(2, 5, (6, 7)^{+}, 1, 3)^{+}, 4$	
Host generator: $2, 5, (6, 7)^{+}, 4$	
Simulated host: $((2, (10, 12)*, 11, 5, 6, 7, (8, 9)*)*, 1, 3)^{+}, 4$	

Notation: * 0 or more, $^{+}$ 1 or more, **()** group, **[]** optional

Because we are using a DES, the time at which the events take place is no longer the wall-clock time but a simulated time. The DES generates the simulated time that is propagated to the BOINC server through messages. The server's daemons synchronize with the new time after EmBOINC wakes them up with signals.

3.3 EmBOINC Settings

Among its several features, BOINC has an administrative interface that project administrators use to access the BOINC database, add or remove platforms, or check the status of the jobs (for example how many jobs are unsent or in progress, or successfully/unsuccessfully completed, are valid, or invalid). This interface has been extended to allow EmBOINC to easily monitor and control projects.

As in actual BOINC projects, EmBOINC supports different applications running simultaneously. Applications can share the simulated hosts partially or completely. For every application, the associated hosts can have different levels of heterogeneity, errors, availability, and reliability. The configuration of an EmBOINC simulation has three parts:

A. The type and number of hosts participating in the project (host modeling).
B. The length, type, number, and other features of jobs generated by each application in the project (job characterization).
C. The BOINC server configuration (BOINC settings).

As shown in Fig. 2, we can predict and analyze different scenarios by changing one or more of these components.

A. **Host modeling**: EmBOINC models a dynamic population of hosts and their attributes. This model can be based on a real BOINC database, extracting and storing useful information in *traces*. As shown in Fig. 3 there is a 1-to-1 relationship between a simulated host and a real host in a database. For

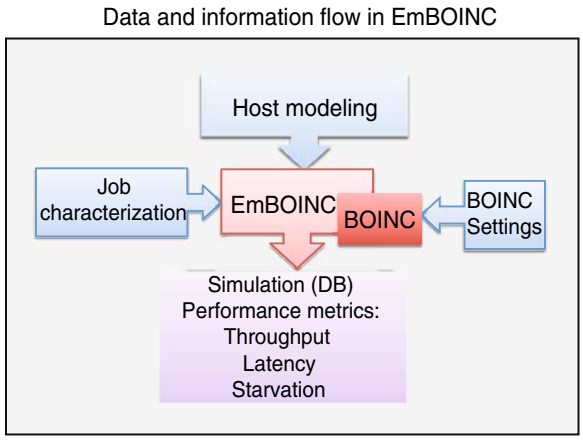


Fig. 2 EmBOINC parameterization

each host, its constant attributes are obtained directly from the BOINC database. Constant attributes include OS, memory, processor speed, available disk space, and bandwidth. Probabilistic attributes are estimated statistically from the host history in the database. These attributes include the error rate and valid rate (uniform probabilities that a host produces an error or a invalid result). Probabilistic attributes also include parameters to model the probability density function (pdf) of the host connection interval. This interval represents the time between two consecutive connections and can be modeled either with a normal distribution (see Eq. 1), where μ is the mean and σ is the standard deviation), or with a Weibull distribution (see Eq. 2). where α is a shape parameter and β is a scale parameter).

$$pdf(x;\mu,\sigma) = \frac{e^{-\left(\frac{(x-\mu)^2}{2\sigma^2}\right)}}{\sigma\sqrt{2\Pi}} \tag{1}$$

$$pdf(x;\alpha,\beta) = \left(\frac{\alpha}{\beta}\right)\left(\frac{x}{\beta}\right)^{\alpha-1} e^{-(x/\beta)^{\alpha}} \tag{2}$$

The EmBOINC user can select which distribution to use. In this paper, we selected the Weibull distribution because it is supported by work of Wolski et al. [25] and our previous work [9], in which we modeled hosts for Predictor@Home and WCG BOINC projects.

B. Job characterization: EmBOINC allows the user to specify heterogeneous or homogeneous workloads for each application. Jobs can have different length, number of replicas, quorum (number of results needed for the validation), sensitivity (impact of operating system and architecture on the results of two instances of the same job), and available platforms, for example 32 or 64 bits, GPU or CPU, or specific operating system. Job characterization can be extracted from a BOINC database or can be specified by the user.

C. BOINC settings: EmBOINC uses the BOINC settings such as the size of the job cache, homogeneous redundancy level, maximum number of job instances assigned per request, and the selection of the scheduling policy.

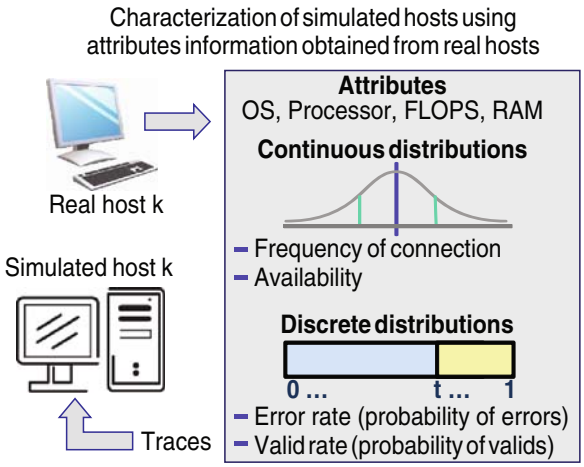


Fig. 3 Statistical host modeling

3.4 Performance Metrics

EmBOINC measures the following types of performance metrics:

- Throughput-based metrics include *project throughput* (rate of valid jobs for the project as a whole) and *application throughput* (rate of valid jobs per application).
- Latency-based metrics include distribution latency (average time from the job generation to the job instance distribution), in-progress latency (average time from job distribution to the reporting of the completed job), execution

Table 2 Characterization of the traces used in the validation

Project	Length in days	Number of hosts	Mean and Std size of jobs (FLOPS)	Error rate	Number of jobs
FightAIDS	18	61460	$\mu = 5.2e13, \sigma = 1.6e12$	$< 5\%$	600129
H.P. Folding II	18	122632	$\mu = 4.6e13, \sigma = 1.1e12$	$< 5\%$	118567

latency (average time the job is executed on the host), and validation latency (average time from the result collection to its validation).
– Starvation-based metrics measure the capability of the BOINC server to keep the volunteer hosts busy. Host starvation can occur when: (1) no work is available; (2) the host lacks sufficient resources (such as memory and disk) to handle any jobs in the servers cache (3) all jobs in the servers cache are committed to a homogeneous redundancy class other than that of the host.

EmBOINC collects the performance metrics during and at the end of the simulation. Final evaluations are stored in log files and can be accessed through the EmBOINC interface. These metrics are also used in the rest of this paper for validation and prediction of BOINC projects.

4 Validating EmBOINC Accuracy

To validate the accuracy of EmBOINC, we used traces from two World Community Grid[3] applications: FightAIDS@Home and Human Proteome Folding II. FightAIDS@Home represents a typical BOINC application in terms of the number of replicas, or instances per job, and the quorum: the average number of replicas and quorum are 3 instances per job. Human Proteome Folding II uses a higher level of replication: up to 19 instances per job, and quorum: up to 14 instances.

Traces were used to describe the simulated hosts and their statistical behavior, as well as the workload characterization of the project. The traces were taken from October 11 to October 29, 2008. During this period the two applications were running in parallel with more than 120,000 active hosts, 46,000 users, and 1,500,000 jobs.

Table 2 summarizes the main features of the traces in terms of length (number of days), size of the host community (number of hosts), job size (FLOPS), host error rates, and number of jobs per traces. Figure 4 shows the distribution of operating systems (OS) and processor vendors for the traces used. As shown in the figure, a large majority of computers participating in World Community Grid are Windows/Intel.

We validated EmBOINC in terms of throughput-based metrics (jobs distributed and collected) and latency-based metrics (distribution and in-progress latency). A validation in terms of starvation was not feasible because BOINC servers do not store this information in their database and therefore this data is not available for comparison.

With reference to the throughput-based metrics, we first compared the cumulative number of job instances distributed and collected per day in EmBOINC simulations versus the same data from the WCG applications. Figure 5a and b compare the cumulative number of EmBOINC simulations versus FightAIDS@Home. Figure 6a and b compare the same cumulative values for the second application, Human Proteome Folding II. As we can see in Figs. 5 and 6, EmBOINC can track and closely follow both the changes in number of jobs distributed per day and jobs collected per day. A comparison in terms of total throughput at the end

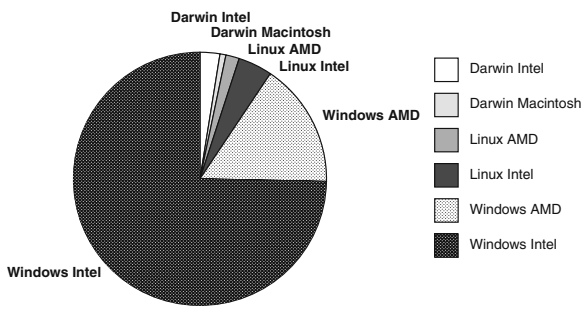
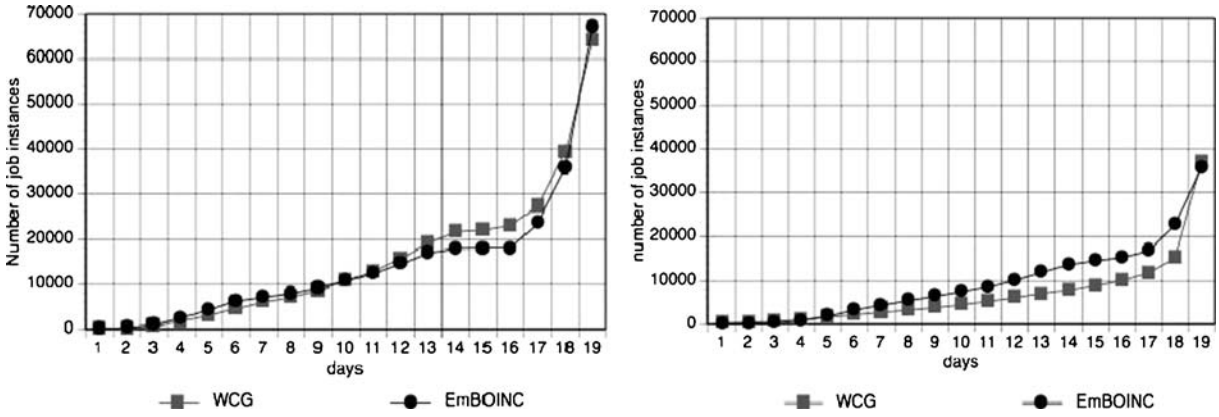


Fig. 4 Characterization of WCG hosts

[3] http://www.worldcommunitygrid.org

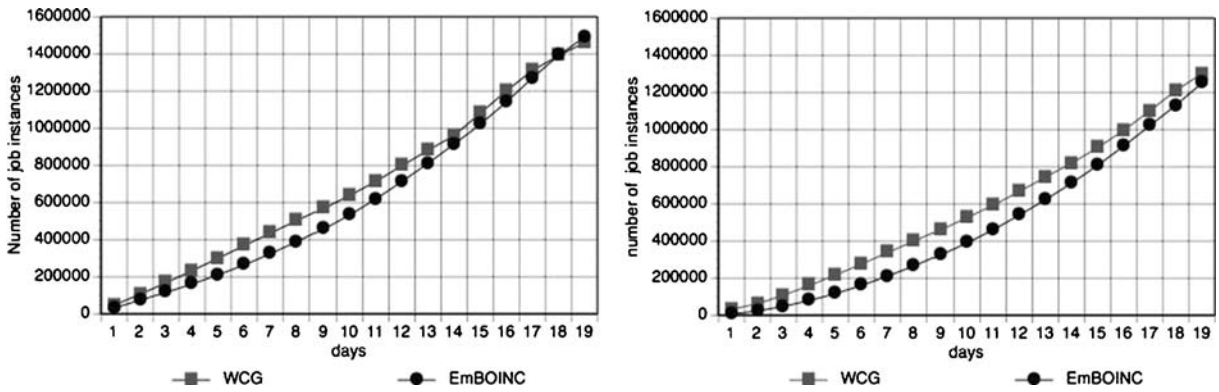


(a) Cumulative job instance distribution (b) Cumulative job result collection

Fig. 5 Job instance distribution and collection for WCG and EmBOINC in FightAIDS@Home (**a**, **b**)

of the 19 days is presented in Table 3. In the table, job instances are classified by: total instances generated (*Total*), instances in progress (*In-progress*), and instances completed (*Over*). The instances that are completed can be categorized as successfully completed without error (*Success*), completed with error (*Error*), and timed-out (*Timed-out*). The successfully completed instances can be categorized as valid (*Valid*), invalid (*Invalid*), and still waiting for validation (*Pending*). Overall, the table shows that the EmBOINC and WCG throughput agree fairly closely.

With reference to the latency-based metrics, Figs. 7 and 8 show the comparison of the distribution and in-progress latencies for World Community Grid versus EmBOINC simulations. In particular, Fig. 7a compares the distribution latency and Fig. 7b compares the in-progress latency of FightAIDS@Home and EmBOINC. Figure 8a and b compare the same latencies for Human Proteome Folding II. The box plot data graphics consist of seven different pieces of information. The whiskers on the bottom extend from the 10th percentile (bottom decile) and top 90th percentile (top decile). Outliers are placed at the end of the top decile whiskers (outliers caps). The top, bottom, and line through the middle of the box correspond to the 75th percentile (top), 25th per-

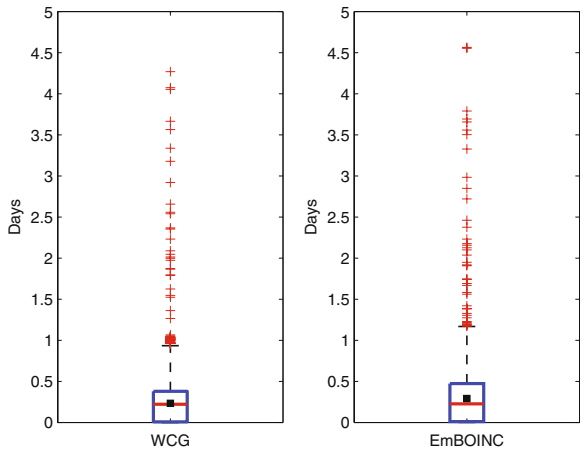


(a) Cumulative job instance distribution (b) Cumulative job result collection

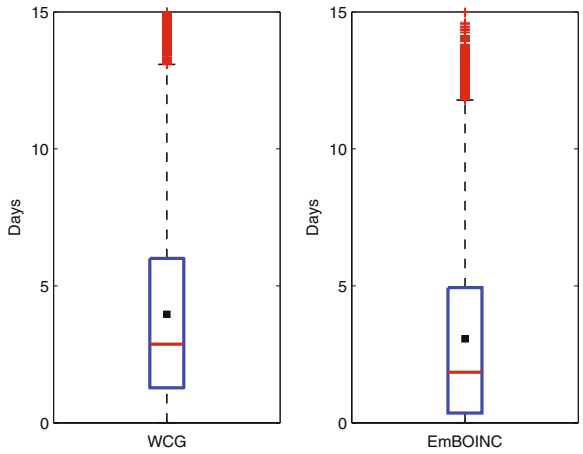
Fig. 6 Job instance distribution and collection for WCG and EmBOINC in Human Proteome Folding II (**a**, **b**)

Table 3 Total throughput values in WCG and EmBOINC simulations

Job state	FAIDS@Home		H.Prot. Folding II	
	WCG	EmBOINC	WCG	EmBOINC
Total	64119	67091	1462602	1491921
In-progress	27032	31404	174381	244185
Over	37087	35687	1288221	1247736
Success	26165	23409	1215593	1181694
Valid	25206	22590	1200421	1167482
Invalid	78	92	14978	14021
Pending	630	479	14	22
Error	1775	1986	57988	53534
Timed-out	9141	10273	14594	12411

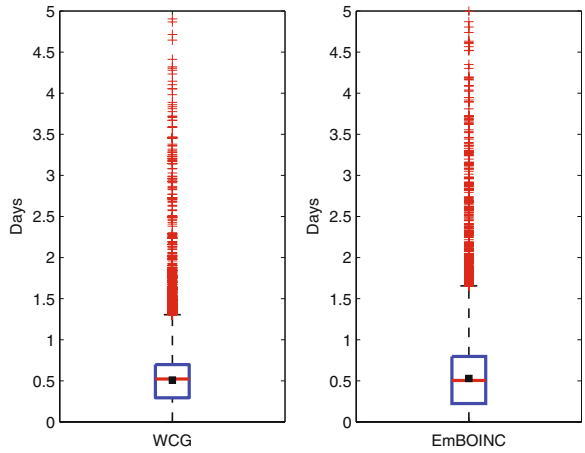


(a) Distribution latency

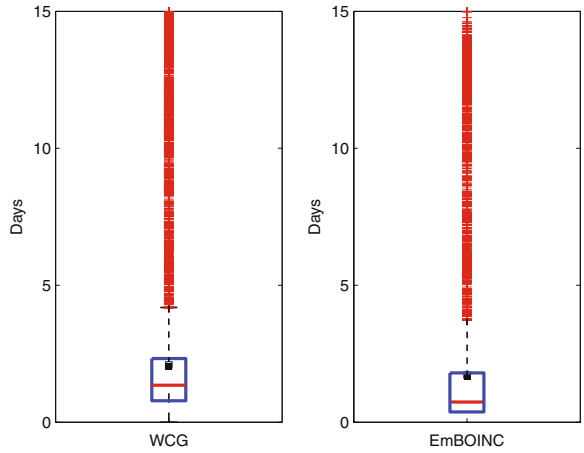


(b) In-progress latency

Fig. 7 Latency comparisons for WCG and EmBOINC in FightAIDS@Home (**a**, **b**)



(a) Distribution latency



(b) In-progress latency

Fig. 8 Latency comparisons for WCG and EmBOINC in Human Proteome Folding II (**a**, **b**)

centile (bottom), and 50th percentile (middle). A square is used to indicate the arithmetic mean. Box plot charts are used in the rest of this paper to effectively display statistical information.

Table 4 shows a quantitative comparison of the distribution and in-progress latencies in WCG and EmBOINC. For a medium-size host community such as that of FightAIDS@Home, EmBOINC can capture both the distribution latency, with an error in the mean of 1.4 hours, and the in-progress latency, with an error in the mean of 18 hours (24%). With larger host communities such as that of Human Proteome Folding II, EmBOINC is still able to capture the distribution latency with an

Table 4 Mean and standard deviation of distribution and in-progress latencies for WCG and EmBOINC

Comparison	Dist. latency (hrs)		In-prog. latency (hrs)	
	Mean	Std	Mean	Std
FightAIDS@Home—WCG	5.52	7.44	93.96	80.35
FightAIDS@Home—EmBOINC	6.96	8.64	75.44	78.72
H. Prot. Folding II—WCG	12.09	6.96	45.12	52.08
H. Prot. Folding II—EmBOINC	12.48	8.16	30.96	49.68

error in the mean of 0.4 hours (15 minutes), but with a slightly higher error for the in-progress latency, with an error in the mean of 15 hours (34%). The size and heterogeneity of the host community, as well as the randomness associated with volunteer computing, are the causes of the higher variability for in-progress latencies.

5 Using EmBOINC: Three Case Studies

BOINC embodies many policies related to the generation, assignment, and validation of jobs. EmBOINC enables the systematic study of such policies. It allows us to predict the performance of new or existing policies, either in the context of a typical existing project or in a hypothetical extreme condition such as highly unreliable or unavailable hosts, extremely large or small jobs, or the presence of many malicious attackers. We now present three case studies illustrating the use of EmBOINC. These studies are based on the Docking@Home project, described briefly in Section 2.3. The Docking@Home database was used to generate the simulated hosts and their statistical behavior. The workload characterization was built synthetically by EmBOINC using the parameters in Table 5. Figure 9 shows the distribution of OSs and processor vendors for hosts in this project.

5.1 Case Study I: Comparison of Job Assignment Policies

A BOINC server's job assignment policy determines the number and type of job instances that are assigned to given host in response to a request for a given number of CPU seconds. We compare the following policies:

- **Policy 0**. This policy enumerates jobs from the server's queue, and assigns them to the requesting host until (based on estimated runtime) the CPU time request has been satisfied. It assigns a job only if the host has sufficient disk space and memory, and only if it is projected to complete the job by its deadline.
- **Policy 1**. This policy is based on Policy 0, but modifies the requested CPU time X as follows: if the host has a recent average credit higher than a certain threshold, then X is increased by 10%; if the error rate of the host is higher than 30%, X is changed in direct proportion to the host availability (fraction of time that the host is on and active) and in inverse proportion to the host error rate.
- **Policy 2**. This policy assigns short jobs to slow hosts and longer jobs to fast hosts.
- **Policy 3**. This policy is based on Policy 0, but modifies the requested CPU time X as follows: if the host has a recent average credit higher than a certain threshold, X is increased by 10%; if the error rate of the host is higher than 5%, X is reduced by half.

We studied the impact of these scheduling policies on the performance metrics. Figure 10a shows the total throughput of the policies. Figure 10b and c show the statistical variation of job instances distributed and collected per day (the higher the better). The distribution and in-progress latencies are shown in Fig. 10d and e respectively (the

Table 5 Main features of the traces used in the predictions

Project	Length in days	Number of hosts	Mean and Std size of jobs (FLOPS)	Error rate	Number of jobs
D@H	25	4396	$\mu = 1.8e13, \sigma = 3.1e13$	10%, 30%, 60%	89052

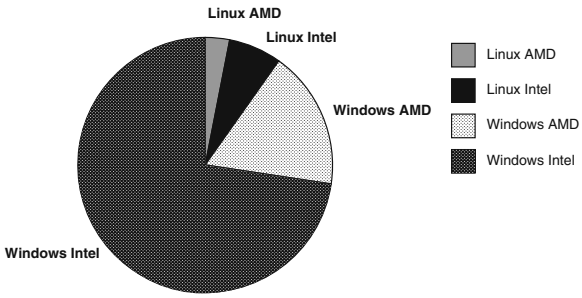


Fig. 9 Characterization of Docking@Home hosts

lower the better). Figure 10f shows the starvation rates (the lower the better). Overall, we can see that Policy 1 outperforms Policy 0 with higher throughput, lower distribution latency, and similar in-progress latency and starvation. The poor performance of Policies 2 and 3 is due to the less lenient approach used in assigning job instances to hosts. In Policy 2 the starvation rate is the highest (Fig. 10f) meaning that the server rejects more host's requests. Also, the poor selection of hosts results in more timed-out results (Fig. 10a). In Policy 3, only those hosts that are highly reliable and available get the amount of work they request; other hosts get only half the work requested. This results in fewer jobs distributed and a larger number of unsent jobs on the server (Fig. 10a).

5.2 Case Study II: Comparison of Varying Homogeneous Redundancy Granularity

The result of a given job may vary when it is executed on hosts with different OSs and architectures [23]. BOINC's homogeneous redundancy (HR) mechanism assigns instances of a job to hosts in the same "numerical equivalence class". The notion of numerical equivalence depends on the application and how it is compiled. BOINC provides three pre-defined equivalence relations, or "granularities":

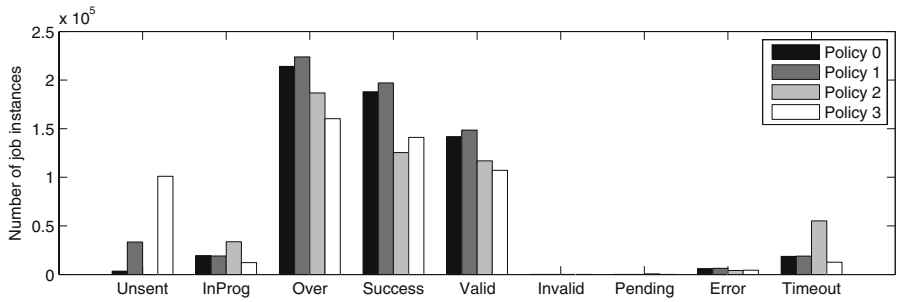
- **HR0 (none):** Instances of a given job are assigned to hosts with no regard to their operating system and architecture.
- **HR1 (coarse):** Instances of a given job are assigned only to hosts with the same operating system.
- **HR2 (fine):** Instances of a given job are sent only to hosts with the same operating system and architecture.

We studied the trade-offs when different granularities of HR are applied to an application whose output depends on both operating system and architecture. Figure 11a shows the total throughput of the policies. Figure 11b and c show the statistical variation of job instances distributed and collected per day (the higher the better). The distribution and in-progress latencies are shown in Fig. 11d and e respectively (the lower the better). Figure 11f shows the starvation rates (the lower the better).

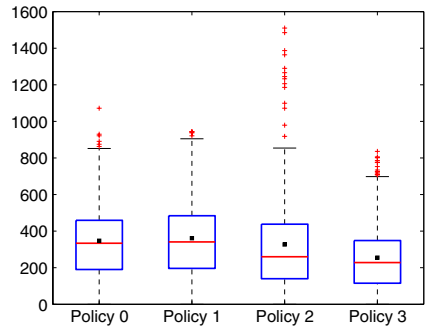
As expected, the stricter the HR level, the more starvation is experienced (Fig. 11f). On the other hand, a more lenient HR results in a higher rate of invalid results (Fig. 11a). In addition, the project administrator must decide how to grant credit for those job instances that are not valid because of improper assignment. Outliers for HR1 and HR2 in Fig. 11d represent those instances that are assigned to rare machines (for example Linux/AMD): these instances tend to spend more time in the shared memory buffer, congesting this buffer and causing further distribution delays for the other instances (Fig. 11d).

5.3 Case Study III: Replication Levels and Error Rates

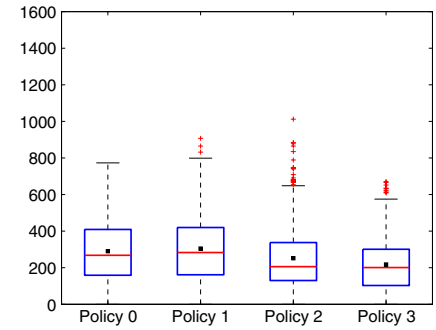
In the context of volunteer computing, jobs fail a nonzero fraction of the time. This fraction depends on both the host population (some hosts may have intermittent hardware failures) and on the application. To deal with high error rates we can consider replicating each job on more hosts. In the third case study we considered three replication levels (2, 3, and 4), and three different error rates (10%, 30%, and 60%). In all nine cases we use the same quorum of two: the server requires only two job instances for validation, but generates and distributes up to four instances. Figure 12a shows the total throughput for the nine scenarios. Figure 12b and c show the number of valid jobs and the number of job instances generated at the end of the EmBOINC simu-



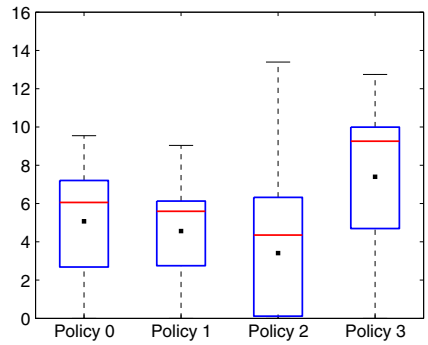
(a) Total throughput



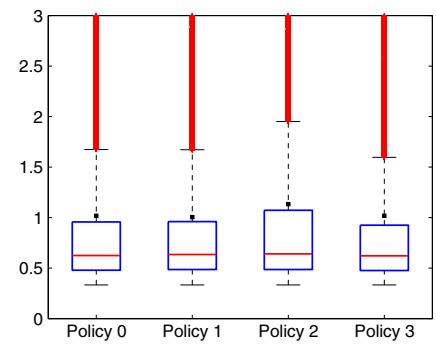
(b) Job instance distribution per hour



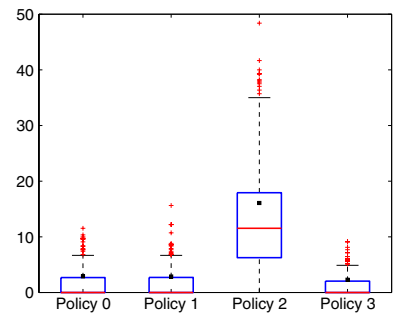
(c) Job instance collection per hour



(d) Distribution latency (days)



(e) In-progress latency (days)

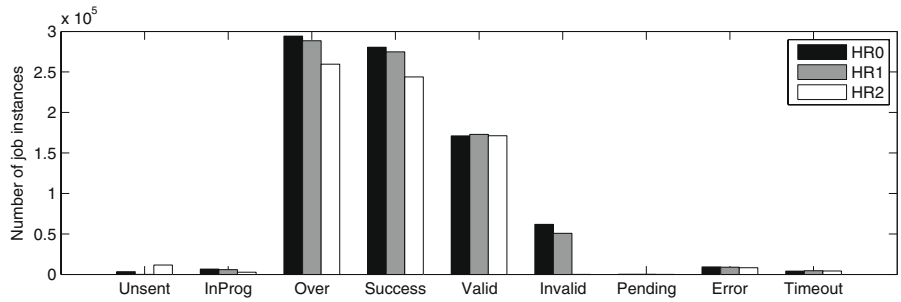


(f) Percentage of starvation

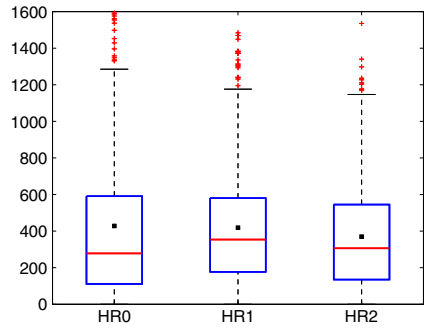
Fig. 10 Case study I: comparison of job assignment policies (**a**–**f**)

lations respectively. Figure 12d and e show the distribution and in progress latencies for the nine scenarios.

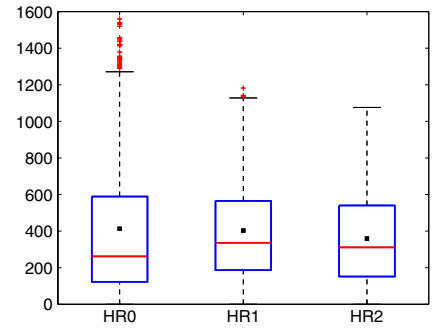
These results show that increasing replication does not compensate for a growing error rate. As shown in Fig. 12a and b, even with the higher rate



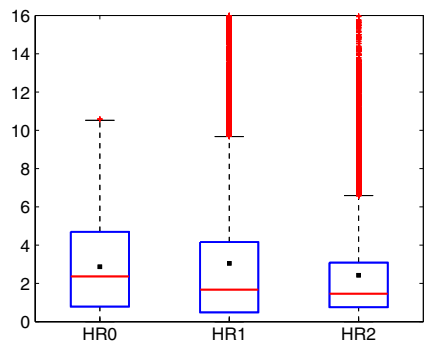
(a) Total throughput



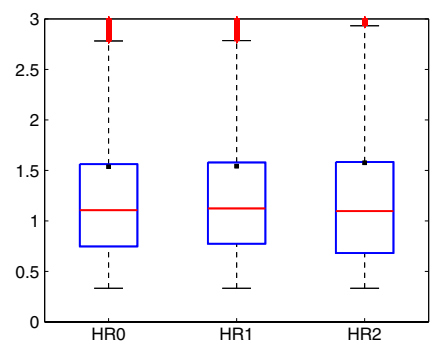
(b) Job instance distribution per hour



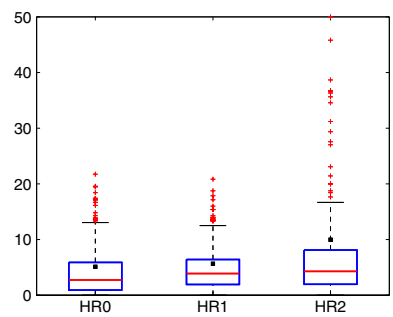
(c) Job instance collection per hour



(d) Distribution latency (days)

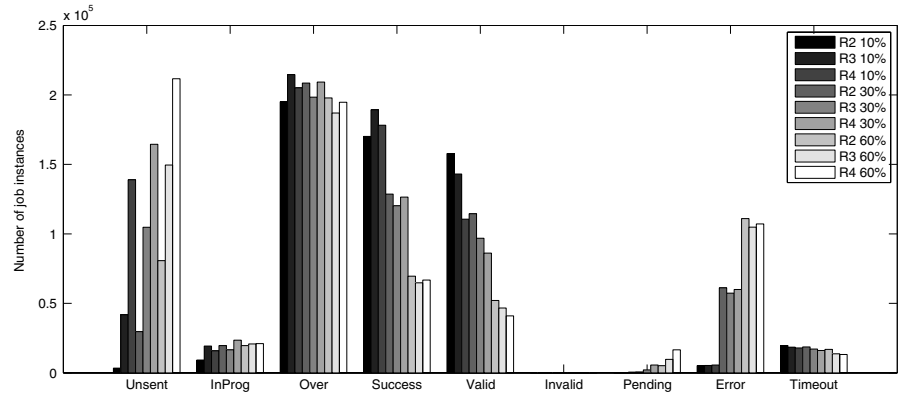


(e) In-progress latency (days)

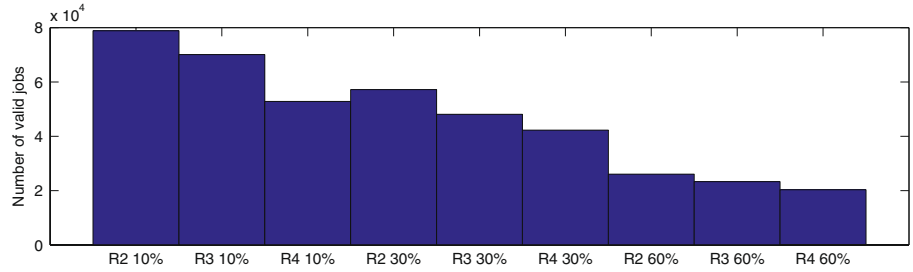


(f) Percentage of starvation

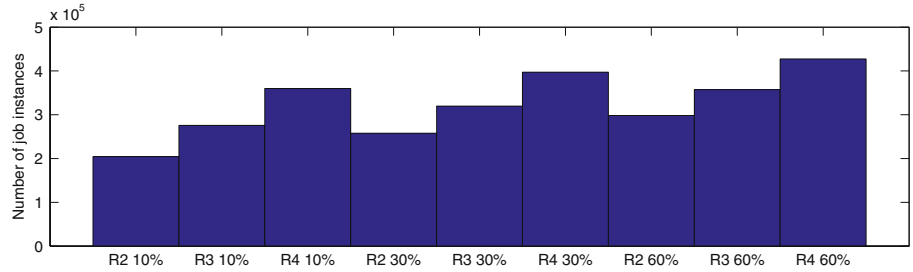
Fig. 11 Case study II: comparison of varying homogeneous redundancy granularity (**a**–**f**)



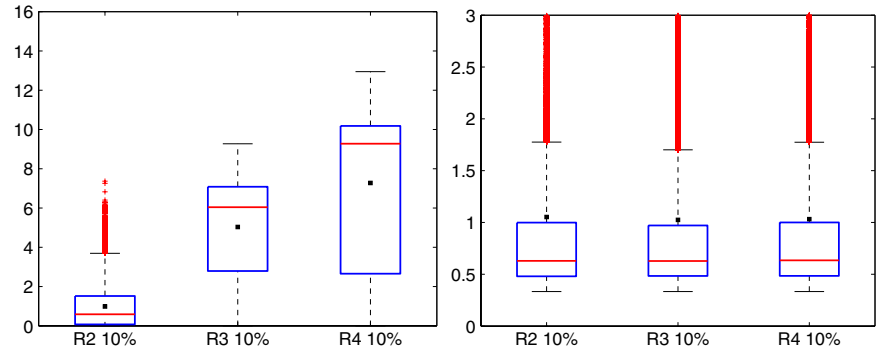
(a) Total throughput



(b) Job instance distribution per hour



(c) Job instance collection per hour



(d) Distribution latency (days) (e) In-progress latency (days)

Fig. 12 Case study III: replication levels and error rates (**a**–**e**)

of 60%, the number of valid results decreases as the replication factor increases. We can observe in Fig. 12c a significant increase in job instances and the associated increase of storage requirements. The BOINC server is using more storage for a longer time because, as showed in Fig. 12d, the

distribution latency grows linearly with the number of replicas. One cause of higher latency is that BOINC does not distribute instances of the same job to the same host, or to hosts belonging to the same user. Hence indiscriminate replication result in a significant waste of computing power. A solution to this problem could be the use of selective replication as proposed in [2].

6 Related Work

In this section we discuss previous work in the modeling and simulation of grid and volunteer computing systems.

6.1 Modeling Grid and VC Projects

Important work has been done to predict distribution and execution times on grid and desktop grid computing systems. Predictions of distribution latencies are presented in [7] where the authors use statistical techniques to predict job queue times in a space-sharing parallel machine with the First-Come-First-Served scheduling policy. In [4, 18, 19], Wolski et al. estimate the time jobs wait in queues by using a non-parametric method and quantiles to define upper time bounds. The prediction of execution times is presented in [22] and [14]. In [22] the authors characterize and predict job execution times with templates derived from execution times of other similar applications. In [14] statistical methods are used to estimate the job execution time (modeled as a random variable) in heterogeneous environments (a large distributed network of heterogeneous machines) supported by an kNN algorithm.

Resource availability is modeled with statistical distributions in [12, 17, 25, 26]. In [12], Hein et al. model unreliable hosts with a Weibull distribution. In [25], Wolski et al. model resource capabilities, dynamic behavior, availability, failure, and connectivity for grid and global computing systems with Weibull and hyper exponential distributions. This work is extended by the authors in [26]. Two- and three-stage hyperexponentials are used to model machine availability in [17] when data shows up-and-down trends. To the best of our knowledge, no past work addresses modeling of distribution latencies and in-progress latencies for VC projects.

6.2 Grid Simulators

The performance of grid applications has been studied in the past using grid simulators such as SimGrid [5] and DGSchedSim [6] or emulators such as MicroGrid [27] and MobiNet [16].

SimGrid [5] is a simulation framework for evaluating cluster, grid, and P2P algorithms. There are important differences between SimGrid and EmBOINC in terms of goals and scope. SimGrid focuses on studying the behavior of resources (for example CPU, disk space, and network) in grid systems accurately. To achieve the high accuracy, SimGrid emulates applications at the cost of reducing its capability to simulate very large environments. DGSchedSim [6] is a trace driven simulator that can be used to evaluate scheduling algorithms of applications executed in heterogeneous desktop grid systems. The main goal of DGSchedSim is to evaluate turnaround time of jobs. Similarly to EmBOINC represents work load based on traces collected from real environments. However, the size of the scenarios that can be simulated in DGSchedSim are several orders of magnitude smaller than what can be simulated with EmBOINC.

Emulators such as MicroGrid [27] and MobiNet [16] focus mainly on emulating the network in Grid environments. They usually inject traffic or capture signals into real LANs or wireless networks to reproduce the network behavior in detail, but do not model the behavior of applications and their jobs.

In general, grid simulators and emulators do not capture the characteristics of BOINC projects since they are not tailored to capture the higher levels of volatility, heterogeneity, and error rates in volunteer computing.

6.3 BOINC Simulators

There are three main BOINC simulators: The BOINC client emulator, SimBOINC, and SimBA. The BOINC client emulator [15] uses the BOINC client code; its goal is to study and replicate the way in which job instances are scheduled and

Table 6 Comparison of BOINC simulators

	EmBOINC	SimBA	BOINC client emulator	SimGrid
Server scheduler	Emulation	Simulation	–	Simulation
Work generation	Emulation	Simulation	–	Simulation
Work collection	Emulation	Simulation	–	Emulation
Work validation	Emulation	Simulation	–	–
Number of hosts	> 100000	< 50000	1	Few 10 000
Host scheduler	–	–	Emulation	Emulation
Host execution	Traces+statistics	Traces+statistics	Simulation	Emulation
Host connection	Traces+statistics	Traces+statistics	Emulation	Simulation
Host availability	Traces+statistics	Traces	Simulation	Emulation
Host reliability	Traces+statistics	Traces	–	–
Number of jobs	> 2*millions*	< 1*million*	N	> 1*million*
Workload	Traces	Input	Settings	Settings
Size of jobs	Traces	Input	Settings	Settings
Replication factor	Traces	Traces	–	–
Number of users	> 20000	–	–	–
Reliability of users	Traces	–	–	–
Number of apps	N	1	N	N

executed inside a single BOINC client. It is not able to model or study server scheduling policies. SimBOINC[4] simulates desktop grids and volunteer computing systems. This simulator is based on the SimGrid toolkit and uses traces obtained from real BOINC clients. Currently this project has been put on hold and does not include recent major changes in BOINC. SimBA [24] simulates the *BOINC server scheduler* and its interaction with a host population. Like EmBOINC, it simulates the generation and distribution of jobs that are executed in a highly volatile, heterogeneous, and distributed VC environment. It also simulates the collection and validation of completed jobs. SimBA uses traces from existing BOINC projects. Unlike EmBOINC, the BOINC server logic is represented abstractly in SimBA's discrete event simulator. This has the disadvantages mentioned earlier: if may not model the server logic accurately, and it is difficult to track the frequent changes to the server code. In comparison, EmBOINC is more robust, accurate, easy to use, and easy to maintain.

Table 6 compares the features available in EmBOINC, SimBA, SimBOINC, and SimGrid. If a feature is marked with a dash, then that feature is not simulated or is not available. The table also shows when a feature is simulated or emulated, and when a feature is characterized by values from traces, modeled statistically, or set explicitly through input values.

7 Conclusions

In this paper we presented EmBOINC, a tool for studying server scheduling policies in BOINC-based volunteer computing projects. EmBOINC uses a hybrid approach in which the part of the system under study (the server) is emulated, while the remainder of the system (the volunteer host population) is simulated.

EmBOINC is easy to install and to use. The selection of host population and parameters, and the analysis of results, can be done through simple Web interfaces. Equally important, EmBOINC is self-maintaining: EmBOINC is integrated directly into BOINC code and can be used with the latest version of the BOINC server software. Its integration in BOINC makes EmBOINC convenient for testing and tuning: every policy tested or tuned with EmBOINC works unchanged with BOINC. The comparison of EmBOINC-generated results with BOINC traces shows its accuracy in predicting the behavior of the real projects.

We have demonstrated the use of EmBOINC to study questions such as: Given an application, what scheduling policy, homogeneous redundancy

[4] http://simboinc.gforge.inria.fr/

granularity, and replication level are best? Given a set of jobs and hosts, what throughput and latency can scientists expect?

EmBOINC is freely available upon request to the authors.

Acknowledgements The authors thank Kevin Reed (IBM) for the World Community Grid traces and the BOINC community for their time, dedication, and computer resources.

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. Anderson, D.P., McLeod VII, J.: Local scheduling for volunteer computing. In: Proc. of the Workshop on Large-Scale, Volatile Desktop Grids (PCGrid) (2007)
4. Brevik, J., Nurmi, D.C., Wolski, R.: Predicting bounds on queuing delay in space-shared computing environments. In: Proc. of the IEEE International Symposium on Workload Characterization (2006)
5. Casanova, H., Legrand, A., Quinson, M.: SimGrid: a generic framework for large-scale distributed experiments. In: Proc. of the 10th International Conference on Computer Modeling and Simulation (UKSIM) (2008)
6. Dominguez, P., Marques, P., Silva, L.: DGSchedSim: a trace-driven simulator to evaluate scheduling algorithms for desktop grid environments. In: Proc. of the Euromicro Conference on Parallel, Distributed, and Network-Based Processing (2006)
7. Downey, A.B.: Predicting queue times on space-sharing parallel computers. In: Proc. of the 11th International Parallel and Distributed Processing Symposium (IPDPS) (1997)
8. Estrada, T., Flores, D., Taufer, M., Teller, P., Kerstens, A., Anderson, D.P.: The effectiveness of threshold-based scheduling policies in BOINC projects. In: Proc. of the 2nd IEEE International Conference in e-Science and Grid Computing (e-Science) (2006)
9. Estrada, T., Taufer, M., Reed, K.: Modeling job lifespan delays in volunteer computing projects. In: Proc. of the 9th IEEE International Symposium on Cluster Computing and Grid (CCGrid) (2009)
10. Estrada, T., Taufer, M., Reed, K., Anderson, D.P.: EmBOINC: an emulator for performance analysis of BOINC projects. In: Proc. of the 3rd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid) (2009)
11. Gathmann, F.O.: Python as a discrete event simulation environment. In: Proc. of the 7th International Python Conference (1998)
12. Heien, E.M., Fujimoto, N., Hagihara, K.: Computing low latency batches with unreliable workers in volunteer computing environments. In: Proc. of the 22nd International Parallel and Distributed Processing Symposium (IPDPS) (2008)
13. Ingalls, R.: Introduction to simulation. In: Proc. of the 2002 Winter Simulation Conference (2002)
14. Iverson, M.A., Ozguner, F., Potter, L.: Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. IEEE Trans. Comput. **48**(12) 1374–1379 (1999)
15. Kondo, D., Anderson, D.P., McLeod VII, J.: Performance evaluation of scheduling policies for volunteer computing. In: Proc. of the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science) (2007)
16. Mahadevan, P., Rodriguez, A., Becker, D., Vahdat, A.: MobiNet: a scalable emulation infrastructure for ad hoc and wireless networks. In: Proc. of the International Conference on Mobile Systems, Applications and Services (2005)
17. Mutka, M.W., Livny, M.: Profiling workstations' available capacity for remote execution. In: Proc. of the 12th International Symposium on Computer Performance Modeling, Measurement and Evaluation (1988)
18. Nurmi, D., Mandal, A., Brevik, J., Koelbel, C., Wolski, R., Kennedy, K.: Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In: Proc. of the International Conference for High Performance Computing, Networking, Storage, and Analysis (2006)
19. Nurmi, D.C., Brevik, J., Wolski, R.: Qbets: queue bounds estimation from time series. In: Proc. of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (2007)
20. Schriber, T.J., Brunner, D.T.: Inside discrete-event simulation software. In: Proc. of the 2003 Winter Simulation Conference (2003)
21. Shannon, R.E.: Introduction to the art and science of simulation. In: Proc. of the 1998 Winter Simulation Conference (1998)
22. Smith, W., Taylor, V., Foster, I.: Using run-time predictions to estimate queue wait times and improve scheduler performance. In: Job Scheduling Strategies for Parallel Processing, pp. 202–219. Springer, New York (1999)
23. Taufer, M., Anderson, D.P., Cicotti, P., Brooks III, C.L.: Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing. In: Proc. of the 14th Heterogeneous Computing Workshop (2005)

24. Taufer, M., An, C., Kerstens, A., Brooks III, C.L.: Predictor@home: a protein structure prediction supercomputer based on global computing. IEEE Trans. Parallel Distrib. Syst. **17**(8), 786–796 (2006)
25. Wolski, R., Nurmi, D., Brevik, J., Casanova, H., Chien, A.: Models and modeling infrastructures for global computational platforms. In: Proc. of the 22nd International Parallel and Distributed Processing Symposium (IPDPS) (2005)
26. Wolski, R., Nurmi, D., Brevik, J.: An analysis of availability distributions in condor. In: Proc. of the 21st International Parallel and Distributed Processing Symposium (IPDPS) (2007)
27. Xia, H., Dail, H., Casanova, H., Chien, A.: The MicroGrid: using emulation to predict application performance in diverse grid network environments. In: Proc. of the Workshop on Challenges of Large Applications in Distributed Environments (2004)