# The Effectiveness of Threshold-based Scheduling Policies in BOINC Projects

Trilce Estrada [1], David A. Flores [1], Michela Taufer [1], Patricia J. Teller [1], Andre Kerstens [1], and David P. Anderson [2]

[1] Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968
*{tpestrada, daflores, mtaufer, pteller, akerstens} @utep.edu*

[2] Space Sciences Lab
University of California at Berkeley
Berkeley, CA *94720*
*daves@ssl.berkeley.edu*

## Abstract

*Several scientific projects use BOINC (Berkeley Open Infrastructure for Network Computing) to perform large-scale simulations using volunteers' computers (workers) across the Internet. In general, the scheduling of tasks in BOINC uses a First-Come-First-Serve policy and no attention is paid to workers' past performance, such as whether or not they have tended to perform tasks promptly and correctly.*

*In this paper we use SimBA, a discrete-event Simulator of BOINC Applications, to study new threshold-based scheduling strategies for BOINC projects that use availability and reliability metrics to classify workers and distribute tasks according to this classification. We show that if availability and reliability thresholds are selected properly, then the workers' throughput of valid results increases significantly in BOINC projects.*

## 1. Introduction

BOINC (Berkeley Open Infrastructure for Network Computing) is a software system for distributed computing using Internet-connected PCs owned by the general public [1]. Several scientific projects use BOINC to do large-scale simulations. In general, the scheduling of tasks in BOINC uses a First-Come-First-Serve (FCFS) policy: a server generates a large number of independent tasks and distributes them to workers in the order in which they are created. No attention is paid to a worker's past performance, such as whether it has tended to perform tasks promptly and correctly.

Our goal is to enhance BOINC scheduling to take into account these worker characteristics. Such new scheduling policies can increase the number of valid results delivered by specified deadlines, while at the same time not increasing the load on the master.

In previous work [2] we proposed the introduction of availability and reliability metrics for the classification of workers in BOINC projects. In this paper we extend this previous work by using these metrics as a part of new scheduling policies that allow us to deliver larger numbers of valid results in shorter amounts of time, as compared to the scheduling policy used in current BOINC projects. The major contributions of this paper are the following:

1. We show that through the use of scheduling policies based on worker availability and reliability thresholds, the workers' throughput of valid results can be changed. Such thresholds can play a key role in assigning tasks only to workers that are likely to deliver valid results by specified deadlines.
2. We show that if worker availability and reliability thresholds are both on either end of the spectrum (i.e., close to 0 or 1), performance is unacceptable, i.e., fewer results are delivered in a given amount of time.
3. We show that BOINC applications have "sweet spots", i.e., worker availability and reliability thresholds that, when used to make scheduling decisions, deliver the largest number of valid results in a given amount of time.

These conclusions are based on the results of trace-driven simulations. Currently, a simulation environment is the best way to evaluate the effectiveness of BOINC scheduling policies. Changing the policy within a running project would involve a large population of BOINC volunteers and could cause them inconvenience.

Our simulator is called SimBA, Simulator of BOINC Applications. SimBA models the behavior of a BOINC master, including its different scheduling strategies for distribution of tasks. The simulator is driven by real traces from in-production BOINC projects such as Predictor@Home (P@H) [3] to model the behavior of workers that join, participate in, and leave a project.

The remainder of the paper is organized as follows. Section 2 briefly discusses background concepts, i.e., Volunteer Computing (VC) environments, BOINC as a representative VC environment, and P@H as a representative BOINC project. Section 3 summarizes the taxonomy of a BOINC project and worker

classifications in terms of worker availability and reliability. Section 4 introduces two scheduling policies for BOINC projects, one of which is a prototype for a new scheduling paradigm. SimBA, the Simulator of BOINC Applications, which is used in this paper to evaluate the effectiveness of scheduling policies, is described in Section 5. Results of the evaluation of the performance of the new scheduling paradigm, in comparison to the one currently used in BOINC, are presented in Section 6. In Section 7 we present related work and in Section 8 we conclude and describe some future work.

## 2. Background

In this section we briefly describe Volunteer Computing (VC) environments, the Berkeley Open Infrastructure for Network Computing (BOINC) as a well-known representative of these environments, and Predictor@Home (P@H), a BOINC project. P@H traces were used to drive the simulations discussed in this paper.

### 2.1. Volunteer Computing

Volunteer Computing (VC) uses computing resources (e.g., desktops and notebooks) connected through the Internet and owned by the public (volunteers) to address fundamental problems in science. VC projects typically are simulations of phenomena in Nature. Some examples of VC projects are: *climateprediction.net* [4], which explores how the Earth's climate may change in the next century under a wide range of different scenarios; *Predictor@Home* [3], which predicts protein structures; and *Folding@Home* [5], which explores the physical processes of protein folding. The VC paradigm emerged in the mid-1990s with two projects, GIMPS and Distributed.net. SETI@Home, launched in 1999, was the first project with appeal that extended beyond hobbyists.

VC environments provide higher throughput than traditional computing systems, such as clusters and supercomputers, at a lower cost in terms of installation, maintenance, power, and infrastructure. However, these environments are particularly challenging because of the nature of their resources: they are volatile, error prone, and heterogeneous.

### 2.2. BOINC and P@H

BOINC (Berkeley Open Infrastructure for Network Computing) is a well-known representative of VC environments. It is an open-source system that harnesses the computing power and storage capacity of thousands or millions of PCs owned by the general public for large-scale 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. Computations may have varying bounds on completion time, and in some cases computations can be aborted. Nevertheless, BOINC projects are very attractive for large-scale simulations because they can potentially sustain a very high processing rate (tens or hundreds of TeraFLOPS). For example, SETI@Home now runs on approximately one million computers, providing a sustained processing rate of about 250 TeraFLOPS [6].

BOINC is based on a master-worker paradigm. Workers, i.e., volunteer computers, request tasks from the master and the master distributes independent tasks among the workers, who send back the computed results when the tasks are completed. BOINC provides a user, i.e., a volunteer, with the ability to attach and simultaneously donate resources to several projects. The user can specify the amount of time that he or she wants to dedicate to each of the projects (e.g., 70% to SETI@Home and 30% to P@H). When a worker successfully sends back a result, it is rewarded with credit (so-called Cobblestones) once the result has passed certain validation checks that aim to determine whether or not the result is valid. A result may be invalid as a result of malicious attacks or hardware malfunctions [7].

Predictor@Home, or P@H, is a BOINC project for large-scale protein structure prediction. The protein structure prediction algorithm in P@H is a multi-step pipeline that includes: (a) a conformational search using a Monte Carlo simulated-annealing approach using MFold [8]; and (b) protein refinement, scoring, and clustering using the CHARMM Molecular Dynamics simulation package [9]. These two applications are our reference applications in this paper.

## 3. Characterization of a BOINC Project

This section briefly presents the taxonomy of tasks in a BOINC project and introduces two metrics for the classification of workers in terms of their availability to compute tasks and reliability to compute trusted results.

### 3.1 Taxonomy of Tasks

Since VC environments are error-prone, BOINC projects use redundant computing to capture possible discrepancies in results due to factors such as malicious attacks or hardware malfunctions. Computation instances of the same task or work-unit (*WU*s) are performed on different PCs and returned results are compared. A BOINC project consists of a set of $N$ work-unit instances (WUIs):

$$BOINC\ project = \{ WUI_k \mid k=0,\ldots,N >> 1\},$$

where $N$ is on the order of thousands or millions.

When VC workers request computational tasks, by contacting the master, the master generates *WU*s and distributes their *WUI*s to the workers in bundles. In-progress *WUI*s, i.e., *WUI*s that have been distributed to workers but for which results have not yet been returned, may be completed successfully or unsuccessfully. Failures to complete successfully may be the result of upload errors, download errors, computational errors, or time-outs (i.e., results are returned to the master with significant delay). Because VC environments are volatile and error-prone, it is unlikely that all distributed *WUI*s complete successfully and all successfully returned results are also valid. The *WUI*s that are associated with one *WU* and have completed successfully are valid if they "agree"; otherwise, they are invalid. The agreement mechanisms vary among different BOINC projects: projects that allow small discrepancies in their results, such as SETI@Home, employ "fuzzy comparisons" while projects with applications that are "divergent", such as P@H, use bit-to-bit comparisons [7].

### 3.2 Worker Availability

In VC environments, in general, and in BOINC environments, in particular, an available worker is not "on" all the time as is the case in cluster or Grid computing. Instead, an available worker is one that is productive, i.e., it returns a large number of results over a certain interval of time. Indeed the worker can be "on" but because of volunteer-imposed restrictions, such as exclusive CPU use by other applications, it may not be dedicated to BOINC projects. Consequently, the **availability** of a worker$_i$ at time $t$ is defined as:

$$availability_i(t) = WUI_{completed\ i}(t) / WUI_{distributed\ i}(t),$$

where $WUI_{completed\ i}$ is the number of work-unit instances completed successfully (and, therefore sent to the master) by worker$_i$ and $WUI_{distributed}$ is the number of work-unit instances that were assigned to worker$_i$. A worker may not return all of its assigned *WUI*s successfully because of errors or timeouts.

### 3.3 Worker Reliability

An available worker is not necessarily a reliable worker: returned results may be affected by hardware malfunctions, incorrect software modifications, or malicious attacks. For example, participants in BOINC projects might increase their CPU frequency to gain more credits. This "overclocking" can cause hardware bit errors (hardware malfunctions) and these errors can affect floating-point calculations but do not crash the computer and might occur sporadically, for example, because of fluctuations in ambient temperature. Such errors can be detected by comparing the results of multiple instances of the same *WU*s. In particular, in BOINC these checks are performed on multiple instances of the same *WU* and for projects such as P@H they are based on strict equality comparison combined with Homogeneous Redundancy (HR) [7]. The **reliability** of a *worker*$_i$ at time $t$ expresses the level of trust for this worker in terms of returned results and is defined as:

$$reliability_i(t) = WUI_{validated\ i}(t) / WUI_{completed\ i}(t),$$

where $WUI_{validated\ i}$ is the number of valid results (instances that have passed the validity check) returned by worker$_i$ over the total number of results or instances completed by worker$_i$.

Availability and reliability are not directly correlated: an available worker is not necessarily reliable. A certain level (ranging from 0 to 1) of availability and reliability characterizes each *worker*$_i$ at time $t$:

$$worker_i(t) \Rightarrow \{availability_i(t), reliability_i(t)\}$$

Our new approach to scheduling in a BOINC project proposes to associate with each worker a level of availability and a level of reliability, and to dynamically update this characterization each time a worker returns results. The scheduling policy described in Section 4.2 takes into account worker availability and reliability when assigning new bundles of *WUI*s to workers: because workers often have discontinuous network connections, when a worker requests computational tasks, it is assigned a bundle of *WUI*s and returns a bundle of results. The number of instances in a bundle is determined by dividing the total number of floating-point operations that the worker is willing to commit to the project by the estimated number of floating-point operations of instances waiting to be distributed.

## 4. Scheduling Policies

Most existing scheduling policies that are used to distribute work-units (*WU*s) in VC environments are based on simple heuristics. In this paper we compare two different policies: the First-Come-First-Serve (FCFS) policy commonly used in BOINC projects and our threshold-based policy. Both policies use Homogeneous Redundancy (HR) for the distribution of work-unit instances (*WUI*s). HR distributes instances of the same *WU* to workers that are computationally equivalent, meaning that they have the same operating system and processor vendor (e.g., Intel or AMD). This yields bit-identical successful results, even for chaotic applications [7].

### 4.1 First-Come-First-Serve Policy

This policy, also called bag of tasks [10, 11], assigns the first available *WUI, usually the oldest,* to the first requesting worker that matches the HR criteria. Although this policy ensures the highest rate of distribution, it assigns tasks to workers that in the past have been shown to lack reliability in terms of results or availability for the project.

### 4.2 Threshold-based Policy

This policy establishes two thresholds, one for worker availability *(threshold$_{availability}$)* and one for worker reliability (*threshold$_{reliability}$*). It uses these thresholds to determine the distribution of *WUI*s to workers; it does not distribute *WUI*s to workers with availability and reliability ratings that are below defined thresholds. These thresholds range between zero and one and are not necessarily equal. For threshold levels of 0 for both availability and reliability, this scheduling policy degrades to the First-Come-First-Serve policy presented in Section 4.1. Worker availability and reliability thresholds can be fixed or can change during a project to better meet the performance goals of the project.

When a worker requests a new bundle of *WUIs*, the master computes the worker's availability and reliability ratings based on the worker's performance history. If both ratings are above the predefined thresholds then the worker is assigned a bundle of *WUI*s. Otherwise, the worker is not assigned work. That is, *worker$_i$(t)* receives *WUI*s, if and only if,

$$availability_i(t) \geq threshold_{availability}$$

and

$$reliability_i(t) \geq threshold_{reliability}$$

This policy has the disadvantage of starving workers that are considered to be unsuitable to perform the work at hand and, thus, virtually removes these workers from a project. To alleviate this situation, a policy could allow workers to improve their ratings. But, this can only be achieved by assigning *WUIs* to these "unsuitable" workers and, thus, permitting them to demonstrate that they are more available or reliable than indicated by their histories. During simulations we observed that the availability and reliability ratings of workers change slowly, therefore, continuing to assign *WUI* bundles to a worker for a time after which the worker is first deemed unsuitable seems to be a reasonable strategy, one with which we currently are experimenting.

## 5. SimBA

The complexity of a dynamic, volatile, error-prone, and highly heterogeneous system, such as a VC environment, can not be accurately represented using deterministic mathematical models, therefore, we use SimBA to study the effectiveness of the two scheduling policies presented in Section 4. SimBA (Simulator of BOINC Applications) is a discrete-event simulator written in Python that models the behavior of a BOINC project. A discrete-event simulator comprises: events, entities, and one or more monitors [12-14]. An event is a condition that occurs at a certain point in time and causes changes in the simulated state. The main events in SimBA are: (a) generate work-unit, (b) generate worker, (c) request instances, (d) generate instances, (e) determine instance output, and (f) check if the simulation is over. The event "determine instance output" determines if an instance was returned successfully or not and, if returned successfully, if it is valid or not. This process follows the rules of BOINC.

An entity is an object able to generate events. In SimBA the entities are: (1) work-unit generator, (2) worker generator, (3) worker, (4) work-unit, and (5) work-unit instance. A monitor collects statistics; the monitor in SimBA is called sampler (S) and it controls the status of the simulator, i.e., in-progress or terminated. In Table 1 we show how the events (letters) are triggered by the entities (numbers) in SimBA. The letters and numbers refer to the events and entities described above. Curly braces indicate more entities generated from an event.

To generate a worker and characterize it, SimBA uses traces from BOINC projects. For each worker, trace information includes: creation time, OS, vendor, life span, average flops, average computation time for an instance, and several rates: unsuccessful rate, valid rate, and timeout rate. The traces used for this paper were collected from the database of P@H during two weeks of execution. Even if traces are used for the generation of the workers, SimBA reproduces the unpredictable behavior of VC environments through some randomness. In particular SimBA uses random distributions for two purposes:

- *To determine the final status of an instance:* SimBA uses a standard uniform distribution to assign the final status of an instance.
- *To simulate the non-dedicated nature of a worker in VC:* SimBA uses a Gaussian distribution to emulate delays in the instance executions due to user interruptions.

| | | |
|---|---|---|
| 1 → a → 4 | 3→ c →{5},3 | 5 → e→ 5 OR no entity |
| 2 → b →3 | 4 → d → {5} | S→f→S |

**Table 1:** Sequences of events (letters) and entities (numbers) in SimBA.

These random mechanisms are driven by a random seed and enable SimBA to behave in a non-deterministic way.

To simulate the assignment of instances to a worker when it requests tasks (i.e., request result event), SimBA models the following scheduling policies:, FCFS, and threshold-based. To run a simulation, SimBA needs the following input parameters:

- worker availability threshold and worker reliability threshold (if these are set to 0, the modeled scheduling policy is FCFS);

- total length of simulated time in hours (SIM_DURATION); temporal resolution of the event queue (SAMPLE_INTERVAL);
- number of *WU*s generated before the first worker is generated (WU_INITIAL_POOL_SIZE); number of *WU*s generated per hour (WUS_PER_HOUR);
- upper limit to the number of instances that can be assigned to a bundle (MAX_TASK_PER_WORKER); and
- time that the master will wait for a result to return (TASK_DEADLINE).

The SimBA output includes the following data: total number of generated and distributed *WUI*s, total number of workers, total number of workers that disconnected from a project during the simulation, and the percentage of successful *WU*s. The output shows how many of the distributed *WUI*s were valid, invalid, in error, timed out, and successful. Among the successful ones the output shows how many were valid and invalid as well as how many were in progress when the simulation ended.

To audit SimBA, we traced its outputs to ensure that no events were lost, events were executed in the correct order, and the number of results is consistent with the number of generated workers and *WUs*. We also compared the output results of six simulator runs (each with different random-number generator seeds) with real P@H output results and observed that, on average, they differ by 5%.

## 6. Evaluation of Simulated Results

We used SimBA to compare the performance of the two scheduling policies described in Section 4, i.e., First-Come-First-Serve (FCFS) and our threshold-based policy, and to understand the behavior, in particular, of the latter. In this section we describe the experimental environment and then present and analyze the simulation results.

### 6.1 SimBA Setup

For all the simulations SimBA parameters were set to the values indicated in Table 2. The selected values are based on the actual configuration of the P@H project. As described in the next section, each experiment had different worker availability and reliability thresholds. SIM_DURATION was set to 340 hours (about two weeks) because our P@H traces cover this interval of time. The P@H traces included 7800 workers.

| SimBA Parameter | Value |
|---|---|
| SIM_DURATION (hours) | 340 |
| SAMPLE_INTERVAL (hours) | 1 |
| WU_INITIAL_POOL_SIZE | 800 |
| WUS_PER_HOUR | 400 |
| MAX_TASK_PER_WORKER | 12 |
| TASK_DEADLINE (hours) | 48 |

**Table 2:** SimBA setup.

### 6.2 Experiments

We ran three sets of experiments for each of the two application traces, i.e., an MFold trace and a CHARMM trace, both taken from the P@H project. For each set of experiments we fixed the random-number generator seed at the beginning of the simulation, making sure that all the simulations are repeatable. A set of experiments consists of 25 simulations, each of which is driven by one of the two application traces. Every experiment in a set is unique in terms of the worker availability and reliability threshold tuple used in the experiment. The value of any of the two thresholds is: 0, 0.25, 0.5, 0.75, or 0.95. The selection of these values was chosen to uniformly sample the tuple search space. For each experiment, we measured the total number of generated, distributed, successful, and unsuccessful results, as well as the total number of valid and invalid results. The results for the simulations driven by the MFold and CHARMM traces are shown in Figures 1 and 2, respectively. Each figure has the same *x*- and *y*-axes, which represent the availability and reliability thresholds, respectively. The *z*-axis indicates our simulation results. Note that for clarity the first two pictures in Figures 1 and 2 have different orientations.

### 6.3 Result Analysis

To facilitate the analysis of the results presented in Figures 1 and 2, we cut them transversely along the main diagonal and we present these results in Table 3. Although the results along the diagonal are not always the best achieved, they show the general tendency that is shown in more detail in Figures 1 and 2. For analytical purposes, in Table 3 we also include the best and the worst results of each experiment. Note that at the end of the 340-hour simulations, besides successful and unsuccessful results, there are *WUI*s still in progress on workers (this explains why the sum of the percentages of successful and unsuccessful results is not 100%). Moreover, we decompose the percentage of successful results into percentages of valid and invalid results (the sum of the percentages of valid and invalid values gives the percentage of successful results).

Values in the table and the two figures are normalized with respect to the total number of generated *WUI*s. This normalization permits us to make a fair comparison of the results across all simulations. This is necessary because the incidence of unsuccessful or invalid results causes the master to generate additional instances of related *WU*s.

Comparing only the values on the main diagonals, Table 3 shows that for both the MFold and CHARMM trace-driven simulations the threshold-based policy with the worker availability and reliability thresholds both set to 0, which is equivalent to the FCFS policy, has the worst behavior. Taking into account all the

results from Figures 1 and 2 we observe that, although FCFS does not always have the worst performance, the worst performance is always associated with a simulation that has one of the thresholds set to 0. This finding reinforces the importance of using the tuple of thresholds.

| Availability/ Reliability Thresholds | Successful Results (%) | Unsuccessful Results (%) | Valid Results (%) | Invalid Results (%) |
|---|---|---|---|---|
| **MFold** | | | | |
| **0.95/0.95** | 83.59 | 6.65 | 81.94 | 1.64 |
| **0.75/0.75** | 85.58 | 5.40 | 83.71 | 1.88 |
| **0.50/0.50** | 85.47 | 5.96 | 83.18 | 2.29 |
| **0.25/0.25** | 85.64 | 6.52 | 83.21 | 2.43 |
| **0/0 (FCFS)** | 80.54 | 13.79 | 76.68 | 3.86 |
| **Best** | 86.83 | 4.52 | 84.40 | 1.52 |
| **Worst** | 77.31 | 15.91 | 75.65 | 4.24 |
| **CHARMM** | | | | |
| **0.95/0.95** | 80.79 | 8.40 | 77.01 | 3.78 |
| **0.75/0.75** | 82.49 | 6.51 | 77.75 | 4.74 |
| **0.50/0.50** | 82.23 | 7.39 | 76.56 | 5.67 |
| **0.25/0.25** | 81.65 | 8.83 | 75.28 | 6.37 |
| **0/0 (FCFS)** | 72.19 | 22.21 | 61.26 | 10.93 |
| **Best** | 85.02 | 4.54 | 78.03 | 3.40 |
| **Worst** | 66.56 | 27.73 | 61.25 | 12.44 |

**Table 3.** Results along the main diagonals in Figures 1 and 2, and best and worst results

Referring to Figures 1 and 2, for both applications, there is a direct correlation between the availability threshold and the number of successful/unsuccessful results. In particular, refer to Figures 1.c and 1.d for MFold and Figures 2.c and 2.d for CHARMM. As can be seen in these figures, the number of successfully returned results increases as the availability threshold increases (ranging from 77% to 86% for MFold in Figure 1.c and from 66% to 85% for CHARMM in Figure 2.c), while the number of unsuccessful results decreases (ranging from 4% to 15% for MFold in Figure 1.d and from 4% to 27% for CHARMM in Figure 2.d). We find a similar relationship between the reliability threshold and the number of valid/invalid results. Referring to Figures 1.e and 2.e, the number of valid results increases with the reliability threshold (ranging from 75% to 84% for MFold and from 61% to 78% for CHARMM), while, as shown in Figures 1.f and 2.f, the number of invalid results decreases (ranging from 1% to 4% for MFold and from 3% to 12% for CHARMM). This shows that by using a threshold-based scheduling policy it is possible to change the number of validated results that can be delivered within a defined interval of time, in our case 340 hours.

In Figures 1.e for MFold and 2.e for CHARMM, we can identify a "sweet spot" where the number of valid results that we can return to the scientists reaches higher values and their differences are within 1%. This spot is delimited by the values of 0.50 and 0.75 for both the availability and reliability thresholds. This shows that when the availability and reliability thresholds are selected properly, the number of trusted results delivered in a given amount of time increases.

In contrast, if we select thresholds at either end of the spectrum (i.e., 0 or 0.95) the simulation results are poor. More specifically, as shown in Figures 1.d and 2.d, if we consider an availability threshold of 0 and a reliability threshold that ranges from 0 to 0.95, we have a very high number of unsuccessful results. As shown in Figures 1.f and 2.f, for an availability threshold that ranges from 0 to 0.95 and a reliability threshold of 0, we have a very high number of invalid results. In both cases, the load on the master is higher because the master reacts to the high number of unsuccessful and invalid results by generating and distributing more instances (Figures 1.a and 1.b for MFold and Figures 2.a and 2.b for CHARMM). For the threshold tuple of 0.95/0.95, the number of trusted results is less than that achieved at the "sweet spot" but higher than that achieved by the other two edge-cases mentioned above. However, as expected, the master reacts to the decreasing number of workers with availability and reliability ratings higher than .95 by generating and distributing less *WUI*s. Accordingly, the search space of the BOINC application reduces and, thus, the probability of finding promising results, i.e., close to results in Nature, reduces.

This confirms our initial claim that when the worker availability and reliability thresholds are both on either end of the spectrum, the performance of the system is poor (i.e., fewer results are delivered in the given amount of time and less sampling is performed).

## 7. Related Work

VC must not be confused with grid computing, which generally means the sharing of computing resources within and between organizations such as universities, research labs, and companies. Differences in the two computing environments result in different scheduling approaches and simulator modeling.

### 7.1 Scheduling Approaches

Scheduling in large-scale computing environments is discussed in [15-17]. These studies primarily target grid environments and, as a result, aspects such as reliability of resources and trustworthiness of application results, which are important when scientific phenomena are simulated on VC environments, are not considered

**MFold Application**

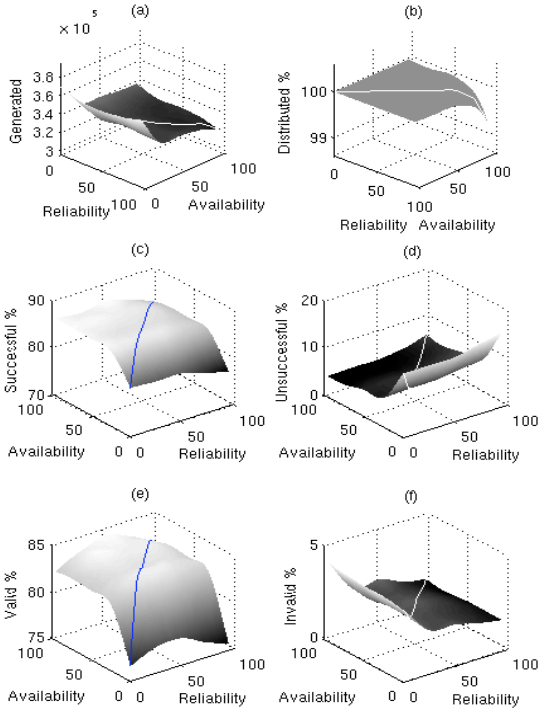


**Figure 1:** SimBA results for MFold

**CHARMM Application**

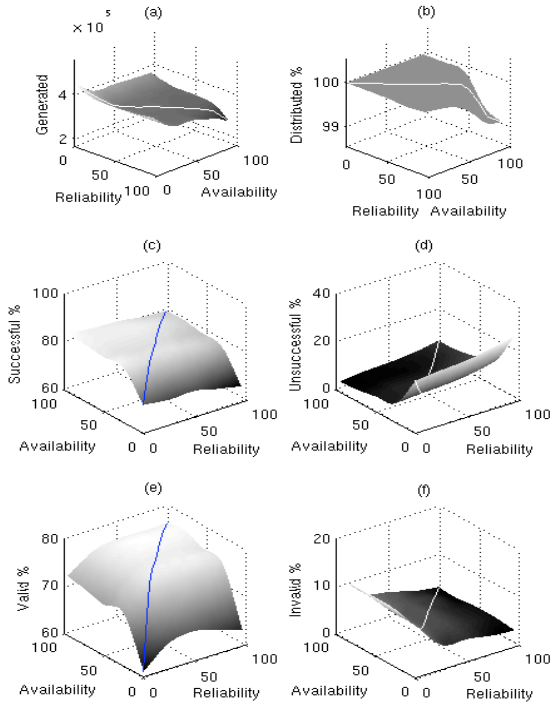


**Figure 2:** SimBA results for CHARMM

Dongarra et al. [18] have done significant work on grid computing and, in particular, have introduced adaptation on GrADS systems. Although GrADS systems are grid environments, they have different features than VC environments. Wolski and others [19, 20] studied the effectiveness of statistical models for predicting machine failure/availability distributions. Statistical models such as exponential, hyper exponential, Pareto, and Weibull distributions were investigated using historical availability, where availability means time during which the machine is powered on. These statistical models are not particularly suitable for VC environments because of server load concerns: the time required by the VC master for accurate predictions of worker availability may be too high.

### 7.2 Simulators

There are several grid simulators (e.g., SimGrid [22], GridSim [23]) that have been used to implement and evaluate scheduling techniques on grid computing environments. The performance of grid applications also has been studied in the past using a grid emulator such as the MicroGrid [24]. In contrast to SimBA, these simulators and emulators do not capture the characteristics of VC since they have not been tailored for this kind of environment.

To our knowledge, little work has been done in the past to address scheduling strategies on VC environments. Probably the closest to the work presented in this paper is the work of Kondo [25]. Kondo built traces using synthetic applications with short task turnaround times (in the order of minutes or seconds). In contrast, our work is based on real traces from an existing BOINC project, P@H, that represent 340 hours of simulation time. The differences in the traces and the models of the computing environments led Kondo to different conclusions than ours. In [25] he claims that the history of nodes does not affect the turnaround time if machines are prioritized and used according to their speeds. In contrast, we show in this paper that in a dynamic environment such as the Internet, the VC environment does not allow an *a priori* determination of worker priority and the history of the workers/machines in terms of availability and reliability plays a key role in determining turnaround time. Also Kondo's definition of availability is different from ours. He considers availability to be the time the machine is "on" and the concept of reliability of results, i.e., trustworthiness, is not addressed in his work, while for us reliability is a key component.

## 8. Conclusions and Future Work

In this paper we demonstrated that by using the proposed availability and reliability metrics to classify workers in a VC environment and adopting a scheduling policy that uses these metrics to determine the distribution of tasks among volunteers' computers, the number of trusted results that can be delivered to scientists can be increased significantly. In particular, for two BOINC applications, MFold and CHARMM, we showed that when using our threshold-based scheduling policy, there are sweet spots, in terms of worker throughput. These sweet spots are associated with availability and reliability thresholds in the range 0.50 to 0.75. When tasks are assigned only to workers that have an availability rating within this range, then, in comparison to the performance of the FCFS scheduling policy used in BOINC, the number of successful results increases, on average, by 14%, with a peak of 18% for CHARMM. When tasks are assigned only to workers that have a reliability rating within this sweet spot, then, in comparison to the performance of the FCFS scheduling policy, the number of valid results increases, on average, by 13%, with a peak of 17% for CHARMM. Note that FCFS ensures the widest range of distribution but also provides the highest frequency of errors.

Using fixed thresholds permanently excludes workers that temporarily fall below the defined thresholds from participating in a BOINC project. Our future work will address this problem by dynamically changing thresholds. This will take into account the fluctuation of the availability and reliability of the whole population of workers and will prioritize replicas waiting for distribution, assigning low-priority replicas to workers with availability and reliability ratings below the thresholds. The slight differences between the sweet spots of the two applications encourage us to further study the sweet spots of a wider range of applications and analyze the correlations between sweet spots and application characteristics.

**Acknowledgments**

This work was supported by the National Science Foundation, grant # SCI-0506429, DAPLDS - a Dynamically Adaptive Protein-Ligand Docking System based on multi-scale modeling.

**References**

[1] DP Anderson: BOINC: A System for Public-Resource Computing and Storage. In *Proc. of the 5th Workshop on Grid Computing (Grid'04),* 2004.

[2] M Taufer, PJ Teller, DP Anderson, and CL Brooks III: Metrics for Effective Resource Management in Global Computing Environments. In *Proc. of the 1st Conference on e-Science and Grid Technologies (eScience'05),* 2005.

[3] M Taufer, C An, A Kerstens, and CL Brooks III: Predictor@Home: A "Protein Structure Prediction Supercomputer" Based on Public-Resource Computing. *IEEE Transactions on Parallel and Distributed Systems.* 2006.

[4] C Christensen, T Aina, and D Stainforth: The Challenge of Volunteer Computing with Lengthy Climate Model Simulations. In *Proc. of the 1st Conference on e-Science and Grid Technologies (eScience'05),* 2005.

[5] V Pande, et al.: Atomistic Protein Folding Simulations on the Submillisecond Time Scale using World Wide Distributed Computing. *Biopolymers*, 2003, 68:91-109.

[6] DP Anderson, and G Fedak: The Computational and Storage Potential of Volunteer Computing. In *Proc. of the Symposium on Cluster Computing and the Grid (CCGrid'06)*, 2006.

[7] M Taufer, D Anderson, P Cicotti, and CL Brooks III: Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular Simulation Results using Public Computing. In *Proc. of the 14th Heterogeneous Computing Workshop (HCW'05),* 2005.

[8] A Kolinski and J Skolnick: Assembly of Protein Structure from Sparse Experimental Data: an Efficient Monte Carlo Model. *Proteins: Structure, Function, and Genetics,* 1998, 32:475-494.

[9] BR Brooks, et al.: CHARMM: a Program for Macromolecular Energy Minimization, and Dynamics Calculations. *J Comp Chem,* 1983, 4:187-217.

[10] D Anderson, E Corpela, and R Walton: High-performance Task Distribution for Volunteer Computing. In *Proc. of the 1st Conference on eScience and Grid Technologies (eScience'05),* 2005.

[11] LB Costa, et al.: MyGrid: a Complete Solution for Running Bag-of-tasks Applications. In *Proc. of the Simposio Brasileiro de Redes de Computadores (SBRC'04)* 2004.

[12] TJ Schriber and DT Brunner: Inside Discrete-Event Simulation Software: How it Works and Why it Matters. In *Proc. of the 2003 Winter Simulation Conference,* 2003.

[13] FO Gathmann: Python as a Discrete Event Simulation Environment. In *Proc. of 7th International Python Conference*, 1998.

[14] RG Ingalls: Introduction to Simulation. In *Proc. of the 2002 Winter Simulation Conference,* 2002.

[15] G Shao, F. Berman, and R. Wolski: Master/Slave Computing on the Grid, In *Proc. of the 9th Heterogeneous Computing Workshop (HCW'00),* 2000.

[16] F Berman, et al.: Adaptive Computing on the Grid using AppLeS. *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, 2003, 14(4):369-382.

[17] E Heymann, MA Senar, E Luque, and M Livny: Adaptive Scheduling for Master-Worker Applications on the Computational Grid. In *Proc. of the 1st Conference on Grid Computing (GRID'00),* 2000.

[18] SS Vadhiyar and JJ Dongarra: Self Adaptivity in Grid Computing. *J of Concurrency Computation: Pract. Exper.* 2004.

[19] J Brevik, D Nurmi, and R Wolski: Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-Peer Systems. In *Proc. of the Symposium on Cluster Computing and the Grid (CCGrid'04)*, 2004

[20] D Kondo, AA Chien, and H Casanova: Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids. In *Proc. of the ACM Conference on High Performance Computing and Networking (SC'04)*, 2004.

[21] H Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proc. of the 1st Symposium on Cluster Computing and the Grid (Grid'01)*, 2001.

[22] A Sulistio, G Poduvaly, R Buyya, and C-K Tham: Constructing a Grid Simulation with Differentiated Network Service using GridSim. In *Proc. of the 6th Int. Conference on Internet Computing (ICOMP'05),* 2005.

[23] H Xia, H Dail, H Casanova and A Chien: 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 (CLADE'04),* 2004.

[24] D Kondo: Scheduling Task Parallel Applications for Rapid Turnaround on Enterprise Desktop Grids. *PhD Dissertation,* UCSD, July 2004.