Probabilistic Allocation of Tasks on Desktop Grids *

Joshua Wingstrom Henri Casanova

Information and Computer Sciences Dept.
University of Hawai'i at Manoa, Honolulu, U.S.A.

Abstract

While desktop grids are attractive platforms for executing parallel applications, their volatile nature has often limited their use to so-called "high-throughput" applications. Checkpointing techniques can enable a broader class of applications. Unfortunately, a volatile host can delay the entire execution for a long period of time. Allocating redundant copies of each task to hosts can alleviate this problem by increasing the likelihood that at least one instance of each application task completes successfully. In this paper we demonstrate that it is possible to use statistical characterizations of host availability to make sound task replication decisions. We find that strategies that exploit such statistical characterizations are effective when compared to alternate approaches. We show that this result holds for real-world host availability data, in spite of only imperfect statistical characterizations.

1 Introduction

Desktop grids are platforms that exploit the idle CPU cycles of distributed and typically individually owned computing resources, which we term *hosts*, such as desktop computers. Many desktop grid systems have enabled large-scale *volunteer computing* on the Internet, as in the famous SETI@home project [16] and others [11][6]. The incentive for host owners to participate in such systems is that the target application is deemed worthwhile. Desktop grids have also been deployed successfully at moderate scale in enterprise environments, e.g., an organization's local area network to execute applications that are of interest to that organization. Several desktop grid infrastructures are available from academia [11, 12, 5] and industry [17]. Although some of these infrastructures vary in usage they all use hosts that are both heterogeneous and volatile. As a result, the overwhelming majority of applications executed on these platforms consist of large numbers (relatively to the number of hosts) of independent tasks, which are often termed "high-throughput" applications.

Several researchers have explored ways of using desktop grids to run non-high-throughput applications. For instance, the MPICH-V project [2] provides a way to execute applications in which tasks can synchronize and communicate using MPI [15]. However, the difficult question of which task should be assigned to which host(s) for best performance remains. This question has been investigated in [7] for applications that depart from the high throughput model due to a small number of tasks (relatively to the number of hosts). The authors empirically show that combining simple host exclusion, host prioritization, and task duplication techniques lead to good application performance. They also attempt to use information about past availability of hosts, but surprisingly find this to be ineffective. However, they employ a naïve approach: they quantify past host availability solely as the average number of useful CPU cycles delivered by each host, when available, in the previous day. The broad question we address in this paper is whether a more precise statistical characterization of host availability is useful to better enable non-high-throughput applications, including ones that go beyond the scenario studied in [7], on desktop grids.

Statistical characterization of desktop grid host availability must rely on availability measurements collected on desktop grids. In this paper we rely on such measurement datasets used in previous work. In particular, two recent articles have reported on and analyzed availability datasets: Nurmi et al. [13] and Kondo et al. [8]. The former provides an in-depth statistical analysis of host availability intervals. These availability intervals are represented as durations, in seconds, between host failures. The latter provides a high-level analysis of CPU availability in terms of actual CPU cycles delivered to a desktop grid application between two application task failures.

The goal of this paper is to determine the value of using a statistical model of host availability for the purpose of task allocation in a desktop grid. A well-known approach is task duplication, but the difficult question is to determine which tasks to duplicate how many times and on which hosts. In

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

this context, our contributions are:

1. A formalization of the relevant task allocation problem and a set of candidate algorithms that use task replication to solve it;

2. An evaluation of the algorithms with an ideal set of assumptions;

3. An evaluation of the algorithms using host availability trace data collected on real-world desktop grids;

4. A demonstration that an in-depth statistical model of failure probability (i) is possible; and (ii) can be useful to improve application performance.

This paper is organized as follows. Section 2 motivates our task allocation problem, states it formally, and describes algorithms to solve it. Section 3 compares the algorithms in ideal "laboratory" conditions. Section 4 compares the algorithms using "real-world" data. Section 5 concludes with a brief summary and discussion of future directions.

2 Redundant Task Allocation

2.1 Problem Motivation and Definition

We consider the problem of executing an application that consists of n identical (in terms of computational requirements) tasks on m hosts. Our goal is to maximize the probability that all tasks run to completion without being interrupted or stopped due to host unavailability. This may seem drastic, as indeed many applications would tolerate some of its tasks to be suspended and later resumed. Although checkpointing guarantees that an application ultimately completes successfully, application makespan may be prohibitively long due to long host unavailability periods (which occur in real-world desktop grids). This problem can be alleviated using task migration, with some overhead. Some desktop grid systems/applications enable task migration while others preclude it.

In this paper we explore solutions to maximize the probability that the application would completed successfully, assuming conservatively that checkpointing or migration are not implemented. The goal is not to guarantee application completion, since this can only be achieved with checkpointing, but instead to minimized the probability of an "application failure" delaying the execution. We term the situation in which one of the application tasks does not complete successfully before the host on which it is executing causes the task to fail an "application failure", with the understanding that checkpointing could then be used to recover from this failure but with perhaps a long delay due to long host unavailability periods. Our approach is thus orthogonal to checkpointing and can thus be easily integrated with a checkpointing/migration infrastructure to guide initial allocation of tasks to hosts so that task checkpointing/resuming/migration is needed only with low probability. We leave a study of the performance/overhead trade-off of such an integration for future work.

The way to improve probability of application completion in our context, as in [7], is then to use task redundancy: multiple replicas of a task can be started on multiple hosts in the hope that at least one of the replicas completes successfully. It is usually straightforward to implement task replication in several applications, e.g., for iterative data-parallel application. Also, transparent task redundancy could be implemented as part of infrastructures like MPICH-V. Task replication in desktop grids is an attractive and realistic proposition because hosts are often plentiful and expendable.

Our task allocation problem is a version of the redundancy allocation problem for a parallel-series system [10], which has been studied extensively in industrial engineering. Our version of the problem is less constrained. An allocation is defined by $(x_i)_{i=1,\ldots,n}$, where $x_i = j$ if an instance of task j is allocated to host i. Our objective is to maximize the probability of success of the entire application. Assume that the probability of task j failing when scheduled on host i is $p_{i,j}$. The probability that all instances of task j fail is:

$$\prod_{i|x_i=j} p_{i,j} \quad (1)$$

Then, the probability that the application executes without failure for a given task allocation is:

$$\mathbf{R}_{sys} = \prod_{j=1}^{n} \left[1 - \prod_{i|x_i=j} p_{i,j} \right] \quad (2)$$

The problem, which we call REDUNDANTTASKALLOC, is to find the allocation $(x_i)_{i=1,\ldots,n}$ that maximizes $\mathbf{R}_{sys}$.

All that is required to specify the above problem fully is a method for estimating the $p_{i,j}$'s. A simple way to model host availability in desktop grids is to consider that each host goes through a sequence of "availability intervals" and "unavailability intervals" [7]. The duration of each interval can be measured as the interval's elapsed time or as the number of operations completed by the host in the interval. Similarly, application tasks are defined by their "size", t, which can denote either a task execution time (on some reference host) or as a number of operations required by the task. Our approach is applicable for both interpretations (as long as they are consistent between interval duration and task size). For now, we simply refer to "durations" and "sizes" with the understanding that they are expressed in the same units.

At a given time, a host is either unavailable, or has been available for some duration since it was unavailable last.

Assuming that availability interval duration can be modeled by a probability distribution ϕ, then $p_{i,j}$ can be defined as:

$$p_{i,j} = \frac{\phi(\gamma_i + t) - \phi(\gamma_i)}{1 - \phi(\gamma_i)}, \quad (3)$$

where γ_i is the duration between the last time host i became available and the time at which the task starts. This approach relies on the existence of ϕ. Fortunately, it was was shown in previous work that it is indeed possible to model availability interval duration reasonably well with standard probability distribution functions [13, 19].

2.2 Algorithms

The allocation problem defined in the previous section is (weakly) NP-hard. We refer the reader to a technical report for the proof [18]. Candidate heuristics to compute the allocation are described below.

EA Algorithm – The EA (Equal Arbitrary) algorithm arbitrarily assigns k or $k+1$ hosts to tasks, where $k = \lfloor \frac{j}{i} \rfloor$. This algorithm does not account for past host availability. It is our baseline from which other algorithms will be judged.

PPB Algorithm – The PPB (Purely Probability Based) algorithm approximates the optimal solution to REDUNDANTTASKALLOC via simulated annealing. We also implemented a dynamic programming algorithm to solve REDUNDANTTASKALLOC exactly. Due to its exponential execution time we could not run this algorithm for desktop grids with more than 14 hosts. For these small grids we found that simulated annealing produces results equal or very close to the optimal.

The PPB algorithm accounts for the time a host has been available (see Eq. 3) to make allocation decisions, which should increase reliability. The question is, does this increase justify the extra work? Indeed, this algorithm requires that host availability measurements be gathered, archived, and processed for off-line statistical modeling.

EPB Algorithm – The EPB (Equal Probability Based) algorithm is a mix of the EA and the PPB algorithms. It approximates the optimal solution to REDUNDANTTASKALLOC in the same manner as the PPB algorithm, but it constrains the solution so that a near equal number of hosts is allocated to all tasks (as the EA algorithm). The EPB algorithm should always perform at least as well as the EA algorithm, even if the statistical model of host availability is far from accurate. And in fact, it could perform better than the PPB algorithm when the statistical model of host availability is not accurate, because in some sense it puts less "trust" into this model.

In practice both the PPB and EPB algorithms require less than 1 second of computation on a 3.2Ghz Intel Xeon processor to converge to a solution for up to problem instances with 400 hosts and 200 tasks. Note that more sophisticated search techniques, such as genetic algorithms, could also be employed and improve the allocations computed by both these algorithms (e.g., as seen in [3] for an industrial engineering parallel series problem).

3 Laboratory Evaluation

3.1 Evaluation Methodology

Problem REDUNDANTTASKALLOC, as it was formulated in Section 2.1, is defined by three parameters (m, n, t) and the probability distribution ϕ. Ideally, we would like to obtain a closed-form expression for the application success probability achieved by all three algorithms. We start by examining the distribution of the probabilities of task failure. This distribution depends on the probability distribution ϕ and the task size t. It was found in [13, 19] that the length of availability intervals in real-world desktop grids can be modeled reasonably well with either a Weibull or a Log-Normal distribution. Unfortunately, finding a closed form solution to the probability of failure distribution based on the parameters to these distributions and the task size seems very challenging.

Fortunately, a little bit of empirical work can help here. If the availability intervals are assumed to be independent, then we observe empirically that the distribution of the probability of failures is perfectly modeled as a *Generalized Doubly Folded Normal Distribution* [14] (GDFN). This was observed for Uniform, Normal, and Log-Normal availability interval duration distributions, and for ranges of parameters for these distributions. We leave a formal proof of this result for future work. This may also be true for correlated data.

The GDFN distribution is defined by the mean (μ) and standard deviation (σ) of the normal distribution on which it is based (with mirror barriers at 0.0 and 1.0). Furthermore, we found that although the nature of distribution ϕ and the magnitude of the task size impact the parameters of this distribution, if the availability intervals are assumed to be independent the distribution of the probability of failure nevertheless remains GDFN. Note that high (resp. low) μ indicates high (resp. low) mean task failure probability, and that high (resp. low) σ indicates high (resp. low) standard deviation of task failure probabilities.

Consequently, we can compare our algorithms using only four parameters: m, n, and the parameters of the GDFN distribution distribution: μ and σ. We have thus eliminated the need for an extensive and labor-intensive evaluation our algorithms for various types of ϕ distribu-

tions and for various parameter values for each type. We have also eliminated the need to evaluate our algorithms for varying values of the task size t.

The EA algorithm always produces the same task allocation for the same values of m and n. Therefore, its performance can be expressed analytically. [4] states that for the parallel series problem the expected mean reliability of the system is:

$$E(\mathbf{R}_{sys}) = \prod_{j=1}^{n} \left[1 - \prod_{i|x_i=j} 1 - \rho_{i,j} \right] , \qquad (4)$$

where $\rho_{i,j}$ it the mean reliability of component j in subsystem i.

In the context of the REDUNDANTTASKALLOC problem, this translates to:

$$E(\mathbf{R}_{sys}) = \prod_{j=1}^{n} \left[1 - \prod_{i|x_i=j} \rho \right] , \qquad (5)$$

where ρ is the mean probability of failure for a given distribution and task size. Now, with Eq. 5, for a given mean probability of task failure, and given n and m, we can compute the expected application success probability when task allocations are computed by the EA algorithm.

It is unlikely that a similar result can be obtained for the PPB and EPB algorithms. Therefore, we must evaluate these algorithms for varying values of our four parameters. We use simulation, generating task failure probabilities from the GDFN distribution. We term the results in this section "Laboratory" because the task failure probabilities are perfectly accurate since sampled from the distribution. For each instance of the problem (i.e., fixed values of m, n, μ and σ) we simulate each algorithm 2,000 times and estimate the achieved probability of application execution success. Our rationale for picking 2,000 repetitions is as follows. We computed the empirical failure rate for a subset of the search space for the EA algorithm for various number of repetitions and found that for more than 400 repetitions, the empirical rate was within 2% of the analytical rate (which, conveniently, can be computed for the EA algorithm). We expect that the same number of repetitions would be appropriate for the other two algorithms, and we conservatively opted for 2,000 repetitions.

3.2 Probability of Success

In this section we use the term *performance* to denote the application success probability achieved by our different algorithms, with high performance meaning high probability.

Figure 1 shows the performance of our algorithms for two fixed values of μ as σ varies. When σ is small, all algorithms perform comparably. This is expected because the probability of task failure is very similar across the hosts. The best solution in this situation is an even allocation, which all algorithms compute. When σ is large, the value of μ does not matter and the performance of each algorithm converges to some fixed value regardless of μ. This is explained by the fact that the GDFN distribution becomes uniform as σ increases. For larger values of σ, there are larger potential gains from using the PPB and EPB algorithms as opposed to the EA algorithm.

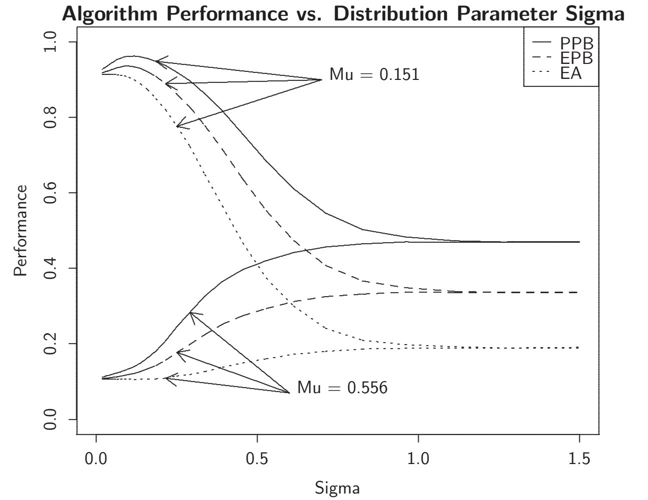


Figure 1. Algorithm performance as a function of of σ for $m = 9$ and $n = 24$. Two sets of curves are plotted for two values of $\mu = 0.151$ and $\mu = 0.556$.

Figure 2 shows algorithm performance for a fixed value of σ as μ varies. All algorithms perform worse as μ increases, which is expected because task failure rates increase. For $\mu = 0.6$ the PPB algorithm generates an allocation that is 9% more likely to succeed than the allocation generated by the EPB algorithm, and 22% more likely to succeed than the allocation generated by the EA algorithm. The EPB algorithm generates an allocation that is $> 5\%$ more likely to succeed than the allocation generated by the EA algorithm, except for when the allocation problems are very easy ($\mu < 0.3$, which corresponds to an average task failure probability $< 34\%$) or very challenging ($\mu > 0.75$, which corresponds to an average task failure probability $> 70\%$).

3.3 Feasible Number of Tasks

While the results in the previous section indicate that the PPB and EPB algorithms outperform the EA algorithm, those results do not highlight the trends with respect to the

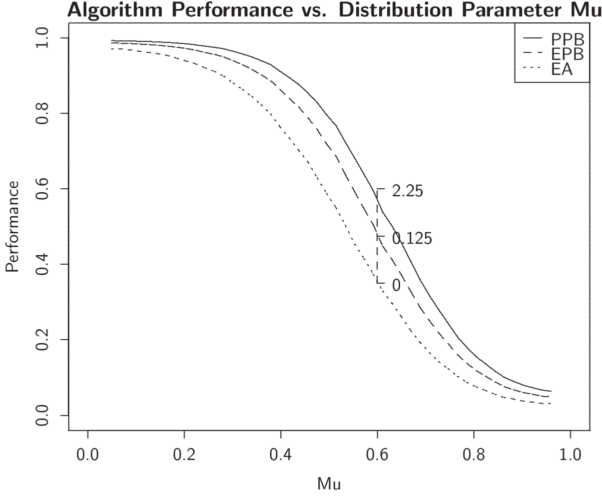


Figure 2. Algorithm performance as a function of of μ for $\sigma = 0.3$, $m = 9$ and $n = 37$.

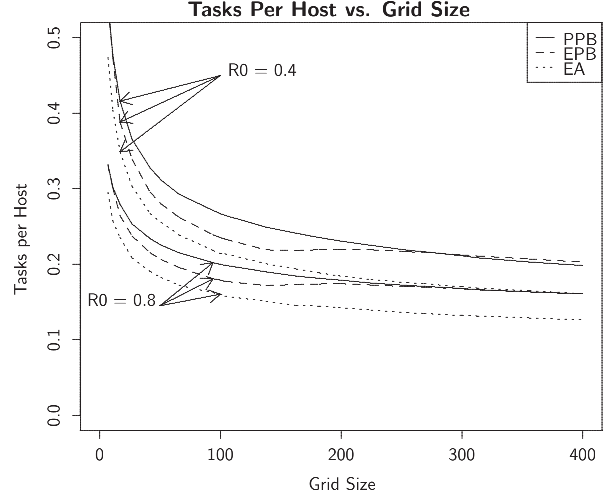


Figure 3. The n/m ratio for a fixed success rate ($\mathbf{R_0}$) as a function of m. $\mathbf{R_0} = 0.4, 0.8$; $\mu = .5$ and $\sigma = .3$.

size of the desktop grid, m. It turns out that there is a glaring weakness of the EA algorithm when m increases. Indeed, the application success probability with the EA algorithm is of the form: $P(\alpha \times m, \alpha \times n) = P(m, n)^\alpha$, where α is a positive real. This is because $P(\alpha \times m, \alpha \times n)$ can be thought of as α dependent versions of the $P(m, n)$ scheduling problem. The probability of success of α versions of the $P(m, n)$ allocation is $P(m, n)^\alpha$. Therefore:

$$P(m, n) = P(mN/n, N)^{n/N} \; . \tag{6}$$

Eq. (6) means that, if the task to host ratio remains fixed, the probability of success decays exponentially as the number of host, m, increases, thus precluding the efficient use of large desktop grids. An important question is then to determine whether the PPB and EPB algorithms suffer from the same weakness.

We define application scale as the number of tasks that can be run in parallel while maintaining a minimum failure rate. Figure 3 plots the average n/m ratio, that is the effective fraction of the desktop grid used to execute the application (ignoring "wasted" hosts due to non-useful replicas), versus m, for our three algorithms. The figure shows two families of curves, each for a different minimum failure rate $\mathbf{R_0} = 0.4$ or $\mathbf{R_0} = 0.8$. We see that, on average, the PPB and EPB algorithms lead to an increase in application scale over the EA algorithm by 24% and 17%, respectively.

Figure 3 also shows that both the EPB and the PPB algorithm exhibit roughly the same performance for large grids. When m is large, the search space for the optimal redundant allocation is enormous. In this case, using simulated annealing, the added constraint on the solution imposed by the EPB algorithm does not preclude reaching a solution that is similar to that reached by the PPB algorithm. As noted earlier, a better search heuristic (e.g., a genetic algorithm) could lead to improved performance.

Based on Figure 3 we conclude that accounting for task failure probability to determine redundant task allocations leads to an increase in application scale when compared to the best probability-oblivious approach, namely the EA algorithm. This result is perhaps not very surprising due to the fact that our experiments in this section were performed in "laboratory" conditions with perfectly accurate task failure probability estimates.

4 Evaluation with Real-World Data

In the previous section we have demonstrated that accounting for host availability for computing a redundant task allocation improves application success probability, meaning that at least one instance of each task completes without being interrupted due to a host failure. However, the experiments in that section were conducted with questionable assumptions. First, our use of the GDFN distribution to generate failure rate probabilities introduces an assumption of statistical independence. (We surmise that this assumption should have little effect because the theory behind why the algorithms work does not rely on statistical independence.) The second assumption is that a perfect statistical model, i.e., a known probability distribution, for the

duration of host availability intervals exists and is known. This assumption does not hold in the real world and approximations have to be used. In this section, we conduct simulations based on host availability trace data collected on real-world desktop grid systems to see if the algorithms that account for host availability still lead to improvements when the above i.i.d. assumptions are violated.

We use host availability trace data collected by other researchers [8]. It is important to note that we only use datasets that quantify availability intervals in terms of number of delivered operations in between failures (as perceived by the desktop grid application), rather than time. The number of delivered operations per interval is not in general proportional to interval duration in seconds (since desktop grid infrastructure can throttle task execution rate to avoid impacting host owners [9]). As a result, availability interval durations in seconds, are not easily connected to task failure probability. Since we are concerned solely with application task failures, we do not make assumptions regarding the homogeneity or the heterogeneity of the desktop grid. Many interesting questions arise when considering the results in this paper in conjunction with checkpointing techniques, and when focusing on the connection between task failure probabilities and application makespan. We leave these questions for future work.

We use two datasets in our experiments: UCB and Entropia. The UCB trace originated from an older data set first reported in [1]. This trace was create from a log generated by a daemon that logged CPU and keyboard/mouse activity every 2s over a 46-day period on 85 hosts. The Entropia trace originated from running the commercial Entropia desktop Grid software on desktop PC's at the San Diego Super Computer Center during a cumulative one month period in the last quarter of 2003. We performed statistical analysis of these datasets, fitted availability interval durations (in terms of operations) to Log-Normal distributions. The results of this fit are summarized in Table 4, which shows for each dataset the parameters of the fitted Log-Normal distribution and the p-values obtained with the Kolmogorov-Smirnov test (see a technical report for all details [19]).

4.1 The SAB Algorithm

We introduce a fourth algorithm that may work well and is straightforward to implement in practice. This algorithm, SAB (Simple Availability Based), is similar to the PPB algorithm, but does not rely on a statistical model of availability interval duration. Instead, it simply assume a linear model for task failure probability, in which the longer the host has been available, the lower the probability of failure:

$$p_{i,j} = 1 - \frac{\gamma_i + t}{C}. \quad (7)$$

where γ_i is the availability duration so far, t is the task size, C is a constant, and $\gamma_i + t \ll C$ for all γ_i and t.

This algorithm is straightforward to implement in practice because it does not require archival and statistical fitting of availability interval durations. We include it in our experiments to evaluate whether in-depth statistical modeling of host availability is truly necessary or whether a simple method to account for host availability is sufficient.

4.2 Results

The input to our scheduling algorithms is the number of hosts to be used m, the number of tasks to execute n, and the fitted availability interval distribution function ϕ as described in Table 4 for each dataset. Figure 4 shows algorithm performance versus the grid size for both datasets, for an n/m ratio of 0.6. As seen in Section 3.3, the performance of all the algorithms decreases as the grid size increases.

Since n and m are discrete, the n/m ratio is only approximately equal to 0.6, causing the "jagged" aspect of the curves. Experiments based on trace data are extremely computationally intensive, thus preventing us from generating a graph with as many data points as in, say, Figure 3.

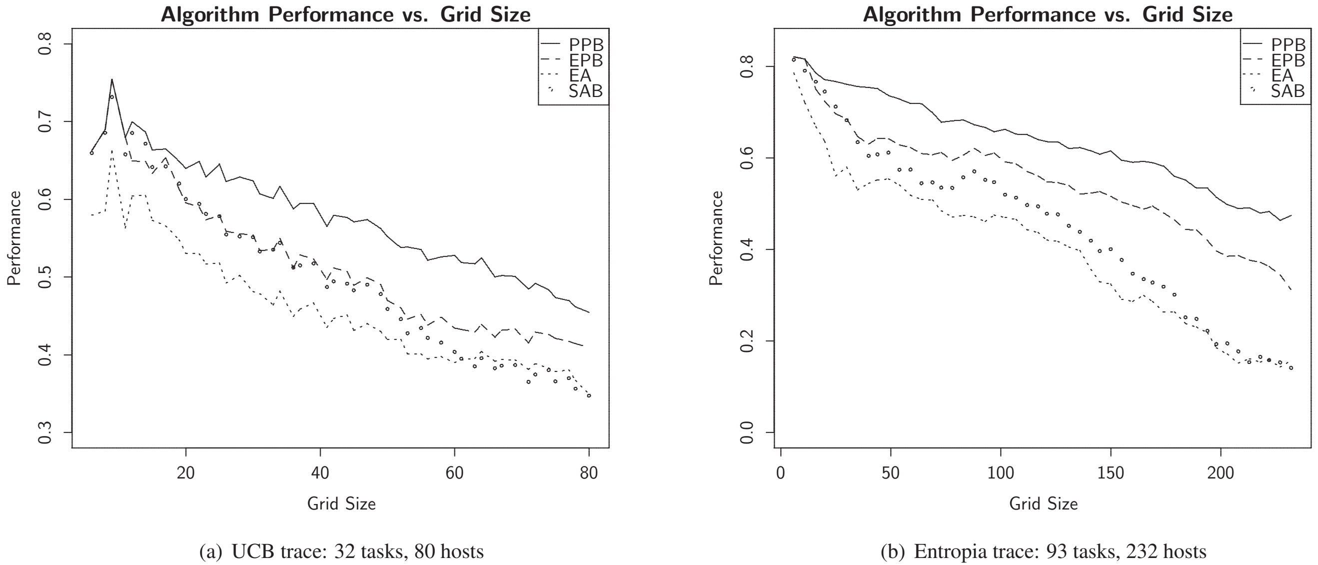
The ranking of the three original algorithms is the same as was observed in the "laboratory" experiments. The crucial result here is that the PPB algorithm achieves the best performance in spite of the i.i.d. assumption being violated in the real-world availability data. Furthermore, the larger the grid size, the better the relative performance of the PPB algorithm when compared to the other algorithms. The SAB algorithm does well for small grid sizes, but its performance becomes similar to that of the EA algorithm for large grid sizes. This important result demonstrates the value of using a more involved approach (i.e., archival of historical availability data, statistical modeling of availability interval durations) to account for host availability for the purpose of redundant task allocation.

5 Conclusion

In this paper we have studied the problem of allocating redundant tasks to desktop grid hosts in a view to maximizing the probability of application completion without task failure. We proposed three algorithms to solve this problem. We first evaluated them in laboratory conditions, that is with i.i.d availability interval durations. We found that significant gains are possible by using a full-fledge statistical model of host availability for computing redundant task allocations. Using two host availability datasets from real-world desktop grids, we confirmed these gains when the i.i.d assumption does not hold.

Table 1. Summary of datasets and their corresponding statistical models.

Name	Number of Hosts	Duration	log-normal fit		KS p-value
			μ	σ	
UCB	80	46-days	16.42	2.419	0.3249
Entropia	232	1-month	18.76	2.188	0.2337



(a) UCB trace: 32 tasks, 80 hosts

(b) Entropia trace: 93 tasks, 232 hosts

Figure 4. Performance of the algorithms as a function of the grid size, using real-world host availability data for UCB $t = 1.0e9$, Entropia $t = 1.0e8$ and $n/m \approx 0.6$

This work can be extended in the following directions. Although our PPB and EPB algorithms used simulated annealing, work in the field of reliability engineering has shown that genetic algorithms may provide a better solution [3]. While not discussed in this paper, better task allocations could be computed by accounting for availability correlation across hosts. An interesting question also is that of availability stationarity and of whether it would be possible to account for it when making task allocation decisions. Studying stationarity however would require availability data collected on real-world desktop grids for long periods of time, which to the best of our knowledge are not readily available. In this paper our performance metric was the probability of successful application completion, defined as an application execution in which at least one instance of each task completed without encountering a failure. This metric is strongly connected to application makespan in a homogeneous system. However, real desktop grids, by their nature, are heterogeneous. An interesting research direction would be to study the problem of computing redundant task allocations that minimize application makespan in heterogeneous systems.

References

[1] R. Arpaci, A. Dusseau, A. Vahdat, L. Liu, T. Anderson, and D. A. Patterson. The Intercation of PArallel and Sequential Workloads on a Network of Workstations. In *Proceedings of SIGMETRICS'95*, pages 267–278, May 1995.

[2] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. Mpichv: Toward a scalable fault tolerant mpi for volatile nodes. In *Proceedings of SC'2002*. IEEE, Nov 2002.

[3] D. Coit and A. Smith. Reliability optimization of series-parallel systems using a genetic algorithm. In *IEEE Transactions on Reliability*, pages 254–260,266, June 1996.

[4] D. Coit and A. Smith. Stochastic formulations of the redundancy allocation problem. 1996.

[5] T. B. O. I. for Network Computing. `http://boinc.berkeley.edu/`.

[6] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *GGF*, 2002.

[7] D. Kondo, A. Chien, and H. Casanova. Resource Management for Short-Lived Applications on Enterprise Desktop Grids. In *Proceedings of SC'2004*, November 2004.

[8] D. Kondo, G. Fedak, F. Cappello, and H. Casanova. On Resource Volatility in Enterprise Desktop Grids. In *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (e-Science 2006)*, December 2006.

[9] D. Kondo, M. Taufer, C. L. Brooks, H. Casanova, and A. Chien. Characterizing and Evaluating Desktop Grids: An Empirical Study. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'04)*, April 2004.

[10] W. Kuo and V. R. Prasad. An annotated overview of system-reliability optimization. In *IEEE Transaction on Reliability*, volume 49, pages 176–187, 2000.

[11] M. Litzkow and M. Livny. Experiences with the Condor disributed batch system. In *IEEE Workshop on Experimental Distributed Systems*, pages 97–101, October 1990.

[12] O. Lodygenski, A. Cordier, G. Fedak, V. Neri, and F. Cappello. "Auger & Xtremweb: Monte Carlo computation on a global computing platform". In *Computing in High Energy and Nuclear Physics*, pages 24–28, March 2003.

[13] D. Nurmi, J. Brevik, and R. Wolski. Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments. In *Proceedings of Euro-Par 2005*, August 2005.

[14] C. Rose. A statistical identity linking folded and censored distributions. In *Journal of Economic Dynamics and Control*, volume 19, pages 1391–1403, 1995.

[15] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI the complete reference*. The MIT Press, 1996.

[16] Sullivan, Werthimer, Bowyer, Cobb, Gedye, and Anderson. A New Major SETI Project based on project SERENDIP data and 100,000 Personal Computers. In *Astronomical and Biochemical Origins and the Search for Life in the Universe*, 1997.

[17] United Devices Inc. `http://www.ud.com/`.

[18] J. Wingstrom and H. Casanova. On the NP-Hardness of the RedundantTaskAlloc Problem. Technical Report ICS2007-11-02, Dept. of Information and Computer Sciences, University of Hawai'i at Manoa, October 2007. `http://navet.ics.hawaii.edu/~casanova/homepage/papers/techreportNP.pdf`.

[19] J. Wingstrom and H. Casanova. Statistical Modeling of Resource Availability in Desktop Grids. Technical Report ICS2007-11-01, Dept. of Information and Computer Sciences, University of Hawai'i at Manoa, October 2007. `http://navet.ics.hawaii.edu/~casanova/homepage/papers/techreportFit.pdf%`.