On Correlated Availability in Internet Distributed Systems

Derrick Kondo, Artur Andrzejak, David P. Anderson

On Correlated Availability in Internet-Distributed Systems

Derrick Kondo
INRIA, France
dkondo@imag.fr

Artur Andrzejak
ZIB, Germany
andrzejak@zib.de

David P. Anderson
UC Berkeley, USA
davea@ssl.berkeley.edu

Abstract

As computer networks rapidly increase in size and speed, Internet-distributed systems such as P2P, volunteer computing, and Grid systems are increasingly common. A precise and accurate characterization of Internet resources is important for the design and evaluation of such Internet-distributed systems, yet our picture of the Internet landscape is not perfectly clear. To improve this picture, we measure and characterize the time dynamics of availability in a large-scale Internet-distributed system with over 110,000 hosts. Our characterization focuses on identifying patterns of correlated availability. We determine scalable and accurate clustering techniques and distance metrics for automatically detecting significant availability patterns. By means of clustering, we identify groups of resources with correlated availability that exhibit similar time effects. Then we show how these correlated clusters of resources can be used to improve resource management for parallel applications in the context of volunteer computing.

1. Introduction

In the past decade, the Internet has grown rapidly in terms of size and bandwidth. This has enabled a new generation of distributed systems spread across the Internet. P2P systems have exploited the Internet for content distribution. Volunteer computing has exploited the free resources in Internet environments for large-scale computation and storage of scientific applications. Computational Grids are beginning to exploit Internet resources as well [12].

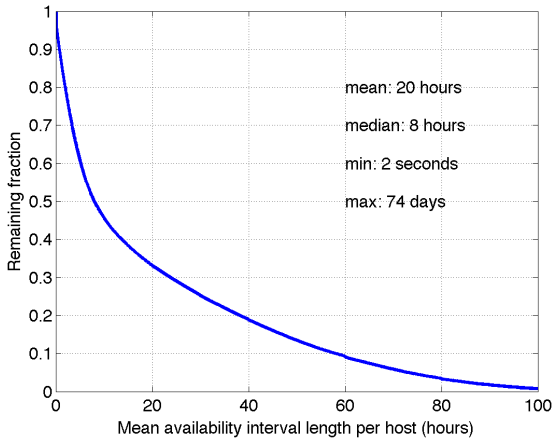
Despite their widespread usage over the Internet, our picture of the Internet landscape is not perfectly clear. Little is known about the time dynamics availability across Internet resources, especially those across residential broadband networks.

This characterization is critical for the effective design and evaluation of Internet-distributed systems. For example, with P2P systems, availability clearly affects the performance of routing and location algorithms. Another example is volunteer computing systems. The time dynamics of availability have strong implications for many aspects of fault-tolerance (such as checkpointing) and resource management (such as scheduling).

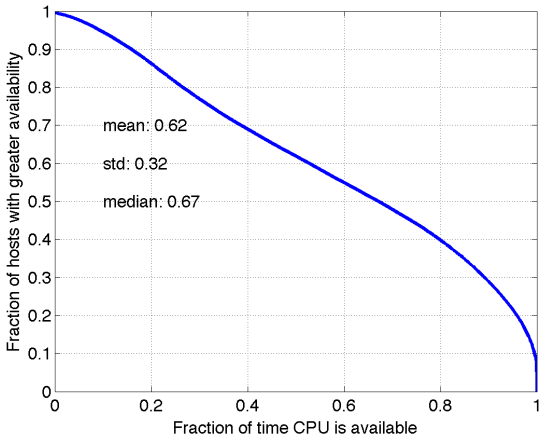
The goal of this study is to measure, observe, and characterize availability, in particular CPU availability, in Internet-distributed systems. Specifically, we provide novel answers to the following questions:

- How do we precisely measure the time dynamics of CPU availability across hundreds of thousands of home desktops?
- How do we scalably and accurately detect patterns of availability among these resources with emphasis on correlated behaviour? Some resources may have unrepetitive and sporadic availability, while others may exhibit repeated patterns. We seek to identify automated methods for separating signals from the noise that scale to hundreds and thousands of resources. We focus on detecting correlated availability as it is an important aspect of characterization exploitable by many Internet distributed systems.
- How can we apply knowledge of these availability patterns for improving resource management of parallel applications and what are the quantitative benefits? For example, knowledge of negatively correlated resources may be valuable for resource management heuristics that must place replicated data or tasks. Identification of positively correlated resources may help enable the execution of complex volunteer computing applications with dependencies, or inter-process communication.

The paper is structured as follows. In Section 2, we describe our measurement method. In Section 3, we present a general characterization of these measurements to gain a basic understanding of the time ef-



(a) CPU interval lengths



(b) Fraction of time CPU is available

Figure 1. CPU availability

fects of CPU availability. (Hereafter, we use the term *availability* synonymously with *CPU availability*.) In Section 4, we delve deeper and investigate patterns of availability. In Section 5, we describe our method and results for detecting patterns using an automated clustering approach. Finally, in Section 6, we apply the discovered patterns for resource management in the context of volunteer computing.

2. Measurement Method

Our approach for gathering measurement data at a large-scale was to use the Berkeley Open Infrastructure for Network Computing (BOINC) [2]. BOINC is a middleware for volunteer computing. The BOINC client currently runs across over 1 million resources over the Internet.

We instrumented the BOINC client to record the start and stop times of CPU availability (independently of which application the BOINC local client scheduler chooses to run). We term an **availability interval** as a period of uninterrupted availability delineated by a CPU start and the next CPU stop as recorded by the BOINC client. The BOINC client would start or stop an interval depending on whether the machine was idle[1]. In this way, we can capture the temporal structure of CPU availability.

This modified BOINC client was then distributed among 112,268 hosts. After a collection period of about seven months between April 1, 2007 and February 12, 2008, the log files of these hosts were collected at the BOINC server for SETI@home [19]. In total, the logs traced 16,293 years of CPU availability. About 81% are at home, 17% are at work, and 2% are at school.

[1]The definition of *idle* is defined by the preferences of the BOINC client set by the user. We assume that the CPU is either 100% available or 0%. Our results in [15], and experience in [17] have shown this to be a good approximation of availability on real platforms.

3. Characterization

In this section, we present a general characterization to quantify the variation in availability among resources.

Figure 1(a) reflects the temporal structure of availability in term of interval lengths. It shows the distribution of the mean interval length of uninterrupted availability calculated per host in terms of a complementary cumulative distribution function (cCDF). The point (x, y) in Figure 1(a) means that y fraction of the clients have mean CPU interval lengths greater than x hours.

The mean and median interval lengths are 20 hours and 8 hours respectively. The mean availability lengths are about 5.25 times greater than in enterprise environments [15]. About 60% of the mean interval lengths are less than 24 hours, indicating the need for fault-tolerance for long-running applications.

Figure 1(b) reflects the volatility of the resources in terms of the fraction of time they are available. It shows the fraction of time the CPU is available in terms of a cCDF. The point (x, y) in Figure 1(b) means that y fraction of the clients are available more than x of the time.

We observe moderate skew. More than 40% of the hosts are available 80% or more of the time. The remaining 60% of hosts have almost a uniform distribution over [0,0.80), making it trivial to model using a least-squares fit. The mean and median fraction of time

available are 62% and 67% respectively. The mean is about 1.3 times lower than that observed in enterprise desktop grids [15]. The standard deviation at 32% is quite high, and is about 50% of the mean. We conclude that residential machines are powered-on for less time (by a factor of 1.3) than those in enterprise environments, but when powered-on, residential machines are more idle (by a factor of 5.25 on average).

Moreover, we wanted to see if there were groups of resources with similar availability and CPU speeds. Figure 2 shows the distribution of hosts binned by speed in terms of maximum floating point operations per second (FPOPS, as determined by the BOINC client) and fraction of time the host's CPU was available.

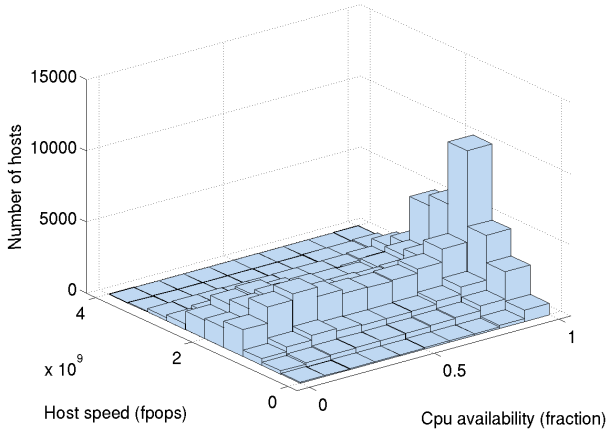


Figure 2. Distribution of host speeds versus CPU availability

We see that nearly 10,000 hosts of speed 2×10^9 FPOPS ($\sim$ 2GHz) are 90-100% available. Another grouping appears near the bar at (2×10^9,0.3), and shows that about 2,500 hosts of speed 2×10^9 have CPU's available 30% of the time.

The data shown in Figure 2 also addresses the relation between resource availability and its CPU speed in FPOPS. We see a whole range of CPU speeds in the 90-100% bin, peaking in the middle. Thus, there is appears to be little correlation between CPU availability and host speed.

Still, it is unclear whether whether availability patterns exist either for a single host over time or across multiple hosts. We look into issue of time effects in the next section.

4. Patterns of Availability

4.1 Time zones

In this section, we examine hour-in-day and day-of-week time dynamics. Our approach is to determine how the fraction of availability varies each hour in the day, or day of the week. As these patterns may be dependent on the local time in each time zone, we consider all the hosts as a whole where the local time is adjusted for time zones, and we consider the hosts only in specific time zones.

In Figure 3(a), we create eight 3-hour slots per day, and determine the total CPU time in each slot over all days and over all hosts. We then normalized this total by the maximum total over all days. In Figure 3(b), we did the same except determined the totals over seven 1-day periods within the week.

We show a plot in Figure 3 for all hosts (after adjusting for time zones), a plot for hosts only in time zone 3600 (corresponding to France, Italy, Belgium, Germany, Spain), and hosts only in time zone -18000 (corresponding to the Eastern United States and Canada, Brazil). These time zones were the top two in terms of the number of hosts.

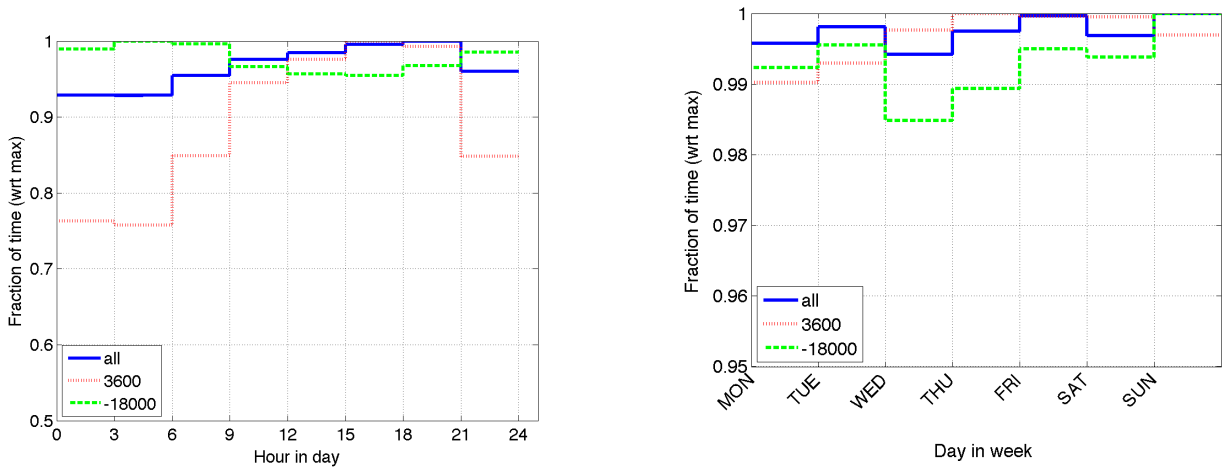
Focusing on the plot for all hosts, we do not observe extreme patterns. For the hour-in-day graph, we see sightly lower availability between 9AM and 9PM. For the day-in-week graphs, we observe little changes throughout the week.

We believed that focusing on smaller groups of hosts by time zone might make patterns more pronounced. So we also observe time effects for each time zone separately, showing the top two in Figures 3(a) and 3(b) for visibility. Potentially, cultural differences among countries in different time zones affect CPU availability.

In time zone 3600 (France, Italy, Belgium, Germany, Spain), hourly fluctuations are more pronounced. We find that between 9PM and 6AM, CPU availability is about 20% less compared to the maximum. By contrast, in time zone -18000 (Eastern United States and Canada, Brazil), CPU time is slightly lower between 9AM and 6PM. One potential explanation is that Europeans tend to turn off machines at night when not in use, where as in other countries such as the US, machines are left on continuously. Still, significant day-in-week time effects are not observed.

Certain time effects could be hidden in Figure 3 because we take the totals over each host grouping. For example, if a large fraction of the hosts are available 100% and contribute most of the compute power, then many patterns could be hidden.

To investigate this issue, we show in Figure 4(a) the fraction of compute power (CPU time multiplied by



(a) Time in day (y range: $[0.50, 1.00]$)

(b) Day of week (y range: $[0.95, 1.00]$)

Figure 3. CPU time effects

FPOPS per host) [2]) contributed by hosts according to their availability levels. The data point (x, y) means that the hosts with CPU availability of x or less contributed y of the total compute power. The plot corresponding to *Uniform* is used as a reference.

We find significant skew. Hosts that are available 90% or more contribute 40% of the total CPU time. Nevertheless, hosts with lesser availability contribute significantly. For example, hosts with intermediate availability between 55% and 90% contribute about 40% of the platform's CPU time. Hosts available 55% of the time or less contribute only 20%.

Also we find that dedicated hosts with 100% availability contribute less than 5% of the compute power. So one cannot assume that dedicated resources provide the majority of the power in the system.

Figure 4(b) shows the same distributions but for each time zone. We observe as much as a 20% difference among the distributions per time zone. For example, hosts in time zone 3600 (Europe) with availability of 60% or less contribute 20% more than hosts in time zone -21600 (US). So more work is done by more available hosts in time zone -21600 compared to hosts in time zone 3600.

Given the significant skew observed, we separated hosts by their availability levels (e.g. 0-10%, 10-20%), and plotted time effects per availability level (to separate hosts that are 100% available for example). The purpose was to to discover more time patterns, especially day-in-week patterns. However, this did not reveal any new patterns.

Nevertheless, we believe that significant patterns exist. So we consider more automated and sensitive methods for detecting such patterns in the next section.

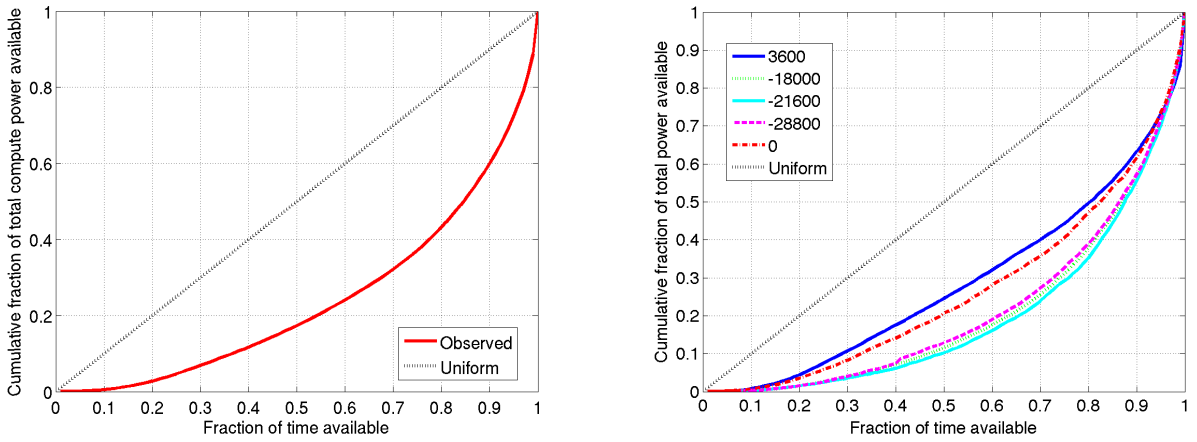
5. Detecting Patterns: A Clustering Approach

Previously, we sought to discover time effects but found few patterns from visualization using simple plots. This was due to the fact that various aspects of the temporal structure of availability intervals could be hidden. For example, when plotting aggregate CPU time per day, negatively correlated patterns of CPU availability could be masked. Our goal is to detect patterns of availability using the following approach that does host-to-host comparisons to construct clusters of resources with similar patterns.

Our approach is to use the k-means algorithm [10] for clustering resources by their availability traces. K-means is a standard clustering algorithm that partitions a data set into k clusters. It does so by iteratively choosing k cluster centers (called *centroids*), calculating the distance of each point in the data set from the k cluster centers, and then grouping the points into each cluster accordingly.

There are several challenges in using such an algorithm. First, we must decide on a representation of the data (in particular the number of dimensions of each point) that allows for scalable but accurate clustering of thousands of data points. Second, we must decide on an appropriate distance metric for the data

[2]Note that when we did not multiply by FPOPS, this did not change the shape of the plot. This supports further the conclusion in Section 3 that CPU speed is not correlated with the fraction of time a host is available.



(a) Cumulative CPU time multiplied by FPOPS, over all hosts (b) Cumulative CPU time per time zone multiplied by FPOPS

Figure 4. Distribution of CPU time contributed

that is sensitive enough to partition the data correctly. Third, we must determine what value of k to supply to the k-means algorithm. That is, we must determine the number of clusters of the data that accurately partitions it. We discuss our main approach for each challenge below, and summarize the alternatives that we considered because of space limitations.

To represent each host trace, we determined the average availability at each hour in the week. This resulted in a 24×7 vector for each host. For each 24×7 vector, we rounded the average availability per hour up to 1 if the value was greater than or equal to 0.75. Otherwise, we it rounded down to 0. By setting the threshold high, we focus only on patterns that are repetitive over a relatively long term.

Other representations we considered included a vector that took into account clock rates. However, this was problematic as relatively slow hosts would be grouped with unavailable hosts. Thus we consider clock rates in a separate step (see Section 6).

To determine the similarity of one host trace relative to another, we used a Hamming distance metric. Given two binary vectors, this metric measures the fraction of unequal values (zeros or ones) in each dimension. Other distance metrics we considered include Euclidean distance. However, we found that Euclidean distance in the context of binary data is not as sensitive as the Hamming distance metric.

To determine the ideal number of clusters, we ran the k-means clustering algorithm for k equal to 30, 20, 10, 5, 4, and 3. We then carefully inspected the quality of each cluster visually by plotting the centroids, and quantitatively by measuring the inter- and intra- vector distances from the centroid. We also evaluate the utility of the clusters for resource management later in Section 6.

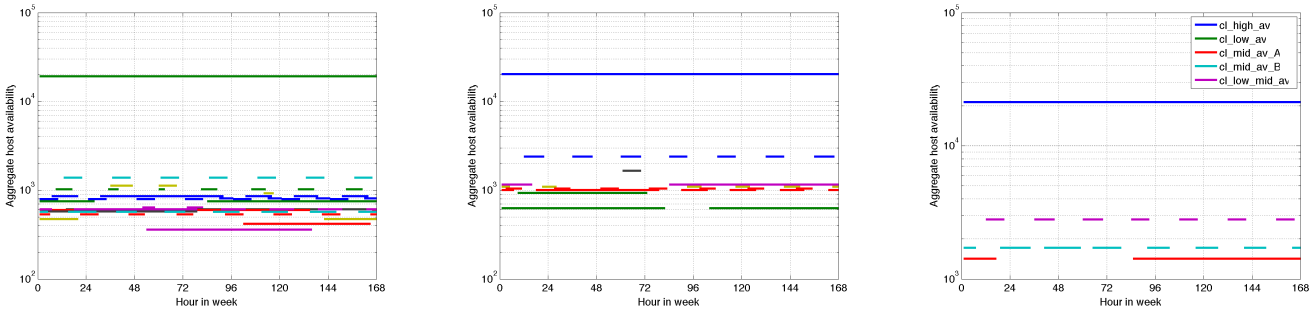
We formed clusters over all hosts, and also hosts in each time zone. Because of space limitations, we only show the clusters over all hosts. Each execution of the k-means algorithm, which was implemented using an interpreted language (Matlab), took less than five minutes on a 1.6GHz Intel Xeon with 4GB of RAM.

Figure 5 plots the centroid of each cluster, multiplied by the number of hosts in each cluster. The Y-value (in log scale) is essentially the number of hosts available in that hour. Note that when availability is 0 the corresponding value in the graph is not plotted to make things more visible[3]. We observe that distinct periodic (negatively correlated) patterns exist, often appearing about 12 hours at a time.

We found the cluster to be best formed for $k = 5$. Table 1 gives the number of hosts per cluster for $k = 5$, the average intra-cluster distance from the centroid, and average availability of the cluster centroid. Table 2 gives the mean, maximum, and minimum pairwise distance among the five centroids. We find that on average the inter-cluster distance is about twice that of the intra-cluster distance.

The top two clusters in size and quality, namely *cl_high_av* and *cl_low_av*, are the groups with 100% and 0% availability respectively. The 100% available centroid has hosts mostly 90-100% available. Few hosts were available 100% of the time. Intra-cluster distance from the centroid is on average between 0-28%. We observed three clusters with similar availability levels

[3]Cluster *cl_low_av* does not appear in Figure 5(c) because it is a vector with only zeros. However, the hosts in this cluster are those with low availability, not necessarily zero availability.



(a) 20 clusters (b) 10 clusters (c) 5 clusters (corresponds to Table 1)

Figure 5. Hosts clustered by availability.

Cluster Name	# hosts	Avg dist	Centroid avail
cl_low_av	22810	0.083	0.000
cl_low_mid_av	2794	0.260	0.429
cl_mid_av_A	1440	0.273	0.601
cl_mid_av_B	1740	0.283	0.643
cl_high_av	21201	0.063	1.000

Table 1. Intra-cluster Statistics (ordered by availability)

Avg	Max	Min
0.585	1	0.400

Table 2. Inter-cluster Statistics (centroid distances)

but opposite correlation patterns (see Figure 5(c)).

6. Applying the Cluster Results

In the previous section, we determined how to automatically, scalably, and accurately identify clusters of availability. But the following question remains: how can applications exploit this information to improve performance?

In this section, we provide an answer to this question in the context of volunteer computing, which uses the free resources in Internet environment for running large-scale scientific applications [19, 16]. Currently, most applications are trivially parallel and compute-intensive. Researchers are trying to broaden the set of deployable applications to include complex tightly-coupled applications [8], especially those with task dependencies.

We claim that the identification of resources with correlated availability is critical for the efficient deployment of these types of applications. We focus on one type of application, namely a parallel application that conducts barrier synchronization periodically.

We determine in simulation the performance when using the clustering method to conduct resource selection. We then compare the performance with that achieved by state-of-the-art resource selection methods. The first method is to simply to select hosts randomly from the entire pool, which is currently done in BOINC [2]. The second method prioritizes hosts first by host availability (between 90-100%) and then by host speed. This second method is similar to what is done in [11, 14]. The scheduling heuristics do not utilize process migration or replication. So if a host fails after resource selection at the beginning of execution, the application waits until the host becomes available again before proceeding with the computation.

For our trace-driven simulation experiments, we simulate an application that does barrier synchronization. The application consists of parallel tasks, where each task is 1-hour in length on a dedicated 2.2GHz system, and conducts a barrier synchronization every 10 minutes.

Over these experiments, we vary the number of hosts, and use the same number of tasks as hosts for the barrier synch application. For a given number of hosts and heuristic, we ran over 3,300 simulations, choosing a unique starting point in the trace for each simulation. Our metric for performance is the average makespan for a given heuristic relative to the best makespan, out of the three heuristics.

With respect to the clustering method, we conducted trace-driven simulation using the hosts in cluster *cl_high_av* of Table 1. Clearly, barrier sync applications not only need correlated availability but also hosts with similar host speeds. Thus we clustered the hosts in cluster *cl_high_av* by clock rates, and used the

largest sub-cluster with FPOPS in the range of 1.7×10^9 ($\sim$1.8GHz)and 2.25×10^9 ($\sim$2.4GHz).

When creating the cluster with k-means, we used data between December 1 and January 31. We used the same 2-month period to determine the availability of hosts for the heuristic that uses prioritization. We ran simulations for the trace data between February 1 and 7, and thus use the clusters as coarse-grain predictions of time effects.

Figure 6 shows the performance of each method relative to the best of the three. We see that using the cluster method is always within factor of 2 of the best, and often at least 1 order of magnitude better than the others. We observed similar gains for the other clusters as well.

The performance of the heuristic that prioritizes hosts by their availability decreases rapidly when the number of hosts (and tasks) reaches 300. After careful inspection of the simulation traces, we found that one particular host was delaying the barrier synchronization because it was almost completely unavailable during the simulation period between February 1 and 7, but available during the previous 2-month training period. In the availability trace of this host, we observed relatively long stretches of unavailability, one of which began near the start of the simulation period.

By contrast, the clustering heuristic excluded this host from its resource pool, placing it in a different cluster with longer stretches of unavailability. The availability traces of those resources in cluster *cl_high_av* exhibited periods of unavailability that were relatively much shorter in duration (versus in longer stretches). Thus, barrier synchronization would only be delayed for a relatively short period of time when using the clustering heuristic.

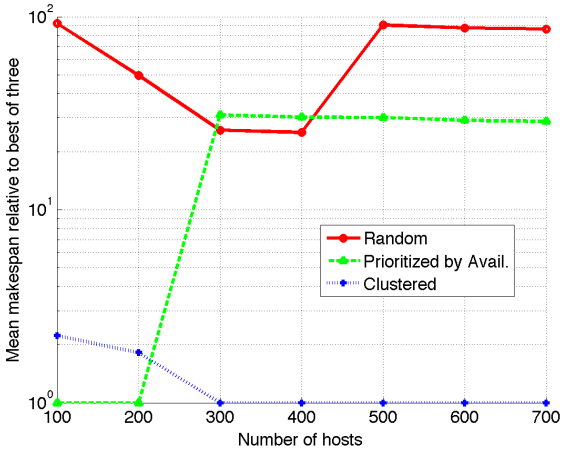


Figure 6. Performance of Barrier Synch Application

7. Related Work

This study differs from other characterization studies in three main respects, namely scale (hundreds of thousands versus only hundreds of hosts), type of hosts characterized (home desktops versus those in the enterprise), and the type of measurements (CPU availability versus host availability).

With respect to scale, the studies in [15, 1, 17, 4] focus only on a few hundred resources. Thus, those studies could not examine the novel issue of detecting significant correlated availability patterns at a large-scale. Compared to [17], our study captures an order of magnitude more traces and is thus less prone to potential bias problems from a relatively small data sample. Other differences between this study and ours are explained below.

With respect to the types of measurements made, the studies in [17, 3] fail to capture the temporal structure CPU availability. In particular, the study [17] runs a BOINC application to gather measurements actively. However, as the BOINC client conducts time-sharing of the CPU among multiple projects, the application may not capture all instances of CPU availability. In [3], the authors do not study the temporal structure of availability.

While there have been a plethora of P2P availability studies [6, 18], these studies focus on *host* availability not *CPU* availability. Clearly, resources can have 100% host availability but 0% CPU availability.

With respect to the types of hosts characterized, the studies in [15, 1, 7] focus on only resources in the enterprise (versus in home environments). As most (81%) of resources in this study were at home, the focus of this study is on residential broadband networks, and our findings are different. For example, the mean availability lengths found in this study are about 5.25 times greater than those in enterprise environments [15]. Also, the mean fraction of time that a host is available is about 1.3 times lower than that observed in enterprise desktop grids [15]. Thus, residential machines are powered-on for less time than those in enterprise environments, but when powered-on, residential machines are less in use. Another is example is that there is considerably more skew in the contribution of volunteers in Internet versus enterprise environments [15] possibly because Internet resources are much more heterogeneous.

8. Summary

We measured, observed, and characterized a large-scale Internet-distributed system. In particular, our contributions were as follows:

- We gathered availability traces from about 110,000 hosts over the Internet, consisting mainly of machines at home. The traces themselves have a broad spectrum of uses for modelling and (trace-driven) simulation. For instance, it would be interesting to evaluate resource management systems designed for large-scale and unreliable environments [11, 9, 13] in the context of this new trace data.
- We gave a general characterization of CPU availability, focusing on time effects. We found that hosts have different availability patterns in different regions of the world. Also, we determined that home machines are powered-on for less time than those in enterprises, but when powered-on, home machines are more idle.
- We determined how to scalably and accurately find correlated time effects using clustering techniques. We use a bit vector representing availability for each the hour of the week, and the Hamming distance as the distance metric.
- By means of this clustering method, we found several novel groups of resources that exhibit similar repeated time effects. These groups are often negatively correlated.
- We then showed how a barrier synchronization application could exploit the clustering results. Performance, shown through simulation experiments, was improved often by an order of magnitude or more relative to state-the-art methods.

In this study, we focused on the discovery of correlated patterns. In other recent work, we focus on the prediction of collective availability [5].

References

[1] A. Acharya, G. Edjlali, and J. Saltz. The Utility of Exploiting Idle Workstations for Parallel Computation. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 225–234, 1997.

[2] D. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, 2004.

[3] D. Anderson and G. Fedak. The Computational and Storage Potential of Volunteer Computing. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, 2006.

[4] A. Andrzejak, P. Domingues, and L. Silva. Predicting Machine Availabilities in Desktop Pools. In *IEEE/IFIP Network Operations and Management Symposium*, pages 225–234, 2006.

[5] A. Andrzejak, D. Kondo, and D. P. Anderson. Ensuring Collective Availability in Volatile Resource Pools via Forecasting. In *Proceedings of the 19th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2008)*, 2008.

[6] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *In Proceedings of IPTPS'03*, 2003.

[7] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed file System Deployed on an Existing Set of Desktop PCs. In *Proceedings of SIGMETRICS*, 2000.

[8] G. Bosilca et al. MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes. In *Proceedings of SC'02*, 2002.

[9] K. Budati, J. Sonnek, A. Chandra, and J. Weissman. Ridge: combining reliability and performance in open grid platforms. In *HPDC*, pages 55–64, 2007.

[10] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, pages 147–153, 2003.

[11] T. Estrada et al. The Effectiveness of Threshold-based Scheduling Policies in BOINC Projects. In *Proceedings of the 2st IEEE International Conference on e-Science and Grid Technologies (eScience 2006)*, December 2006.

[12] Kacsuk, P et al. Enabling Desktop Grids for e-Science. `http://www.edges-grid.eu`.

[13] J.-S. Kim, B. Nam, P. J. Keleher, M. A. Marsh, B. Bhattacharjee, and A. Sussman. Resource discovery techniques in distributed desktop grid environments. In *GRID*, pages 9–16, 2006.

[14] D. Kondo, A. Chien, and C. H. Rapid Application Turnaround on Enterprise Desktop Grids. In *ACM Conference on High Performance Computing and Networking, SC2004*, November 2004.

[15] D. Kondo, M. Taufer, C. 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.

[16] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande. Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2003.

[17] P. Malecot, D. Kondo, and G. Fedak. Xtremlab: A system for characterizing internet desktop grids (abstract). In *in Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, 2006.

[18] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedinsg of MMCN*, January 2002.

[19] W. T. Sullivan et al. A new major SETI project based on Project Serendip data and 100,000 personal computers. In *Proc. of the Fifth Intl. Conf. on Bioastronomy*, 1997.