J. Parallel Distrib. Comput. 72 (2012) 281–296

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Long-term availability prediction for groups of volunteer resources

Daniel Lázaro [a,*], Derrick Kondo [b], Joan Manuel Marquès [c,d]

[a] Internet Interdisciplinary Institute, Universitat Oberta de Catalunya, Barcelona, Spain
[b] INRIA Rhone-Alpes, Laboratoire LIG, France
[c] Computer Science, Multimedia and Telecommunications Studies, Universitat Oberta de Catalunya, Barcelona, Spain
[d] Department of Computer Architecture, Universitat Politècnica de Catalunya, Barcelona, Spain

ARTICLE INFO

Article history:
Received 15 June 2011
Received in revised form
11 October 2011
Accepted 24 October 2011
Available online 30 October 2011

Keywords:
Availability prediction
Reliability
Trace analysis
Volunteer computing
Service deployment

ABSTRACT

Volunteer computing uses the free resources in Internet and Intranet environments for large-scale computation and storage. Currently, 70 applications use over 12 PetaFLOPS of computing power from such platforms. However, these platforms are currently limited to embarrassingly parallel applications. In an effort to broaden the set of applications that can leverage volunteer computing, we focus on the problem of predicting if a group of resources will be continuously available for a relatively long time period. Ensuring the collective availability of volunteer resources is challenging due to their inherent volatility and autonomy. Collective availability is important for enabling parallel applications and workflows on volunteer computing platforms. We evaluate our predictive methods using real availability traces gathered from hundreds of thousands of hosts from the SETI@home volunteer computing project. We show our prediction methods can guarantee reliably the availability of collections of volunteer resources. We show that this is particularly useful for service deployments over volunteer computing environments.

1. Introduction

Context and motivation. Volunteer distributed computing harness the computing power and storage of hundreds of thousands of Internet resources. Currently, volunteer systems provide about 10 PetaFLOPS of computing power for scientific projects. Using volunteers, these projects have produced breakthrough scientific results [26] published in over 60 of the world's most prestigious conferences and journals, such as Science and Nature.

However, the volatility of Internet volunteer resources has limited the range of applications to those that are high-throughput and embarrassingly parallel. Other applications, such as communicating parallel jobs or web services, could potentially benefit from this cost-effective platform, if the volatility of these resources could be masked or managed.

In volunteer computing, machines are controlled by individual users, who may not be able or interested in keeping them continuously connected and available for the system. Therefore, these systems are highly dynamic, and typically offer no guarantees as to how long a given resource will remain available, or how many machines will be available at a given time. To ensure availability, we focus in this work on availability prediction for a *collection* of Internet-distributed non-dedicated resources, like those used in volunteer computing.

Application availability requirements. One example of an application which could benefit from volunteer computing may be deploying a service over a set of hosts. The service is considered available when there are at least N hosts offering it. This notion is called collective availability [3]. The required number of hosts may be determined by the functionality and internal architecture of the service (it needs N hosts to perform its tasks), or by the load of the system (N replicas of the service are required to serve the existing clients). In such a system, the target is to keep the service available for an arbitrarily long period of time, while using the minimum possible amount of resources.

A similar case is that of data storage, where a data object is stored over distributed hosts and needs to be available for clients to retrieve it at any time. This kind of application could also benefit from volunteer computing, as common desktop machines usually have spare disk space [2]. In order to improve the availability of the stored object, it may be replicated: it is stored in M hosts out of which at least one needs to be available at a given moment so that the object can be retrieved. Another possibility is to use erasure codes [11], which divide a data object in M fragments. A client needs to obtain N fragments ($N < M$) to restore the whole file. In order for such a system to work, it is important to minimize the number of replicas or fragments that are stored in order to make an efficient use of both storage and communication

* Correspondence to: IN3, Media-TIC Building, c/Roc Boronat, 117, 7th floor, 08018 Barcelona, Spain.
E-mail addresses: dlazaroi@uoc.edu (D. Lázaro), dkondo@imag.fr (D. Kondo), jmarquesp@uoc.edu (J.M. Marquès).

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.10.007

resources. This case is similar to the service deployment case, as both require at least N available hosts. The main difference is that this only requires host availability (i.e., the host is reachable), while services require CPU availability (i.e., the host's CPU is free to execute the service). We discuss these different availability notions in Section 2.

Previous work. Most of the existing research in availability prediction for volunteer computing [17,3] has focused on predicting short-term availability, i.e., how long will a current availability or unavailability stretch last. The prediction interval for continuous availability for the short-term is typically no greater than 4 h. This is useful for the deployment of individual tasks, the current focus of volunteer computing, to predict if a machine will be available long enough for the task to finish and return its result. For longer tasks, predicting when a node will cease being available may also be useful for scheduling checkpoints or migrations. The only important thing here is accuracy, and the most recent data can be used to increase prediction quality.

However, in the other cases presented (service deployment and data storage), the availability requirements of the application are different, so using the same availability prediction techniques may not be optimal. One way to adapt short-term prediction for achieving long-term availability is dividing the service lifetime (potentially very long) into shorter prediction intervals. Some techniques can be used to predict if a host will be available during this interval, and hosts are selected to deploy the service based on these predictions. After the prediction interval has finished, the most recent data can be used to predict host availability during the next interval and modify the deployment as required.

This technique can give good results for relatively short-lived services [3]. However, most hosts in volunteer environments suffer transient disconnections. Therefore, this adaptation will require migrations at a constant rate in the long run, even when no host permanently leaves the network. This will have a communication cost that may not be negligible.

Approach and contributions. A different approach would be to consider a permanent deployment, where once a service is deployed in a host, the host will execute the service whenever it is available. This breaks the correspondence between disconnections and migrations, as we consider that a departing node will later return (considering permanent departures as a separate case) and execute the service without requiring another migration process (file transfers, etc.). This is often the model used in distributed storage systems [7,19], where data objects can remain in hosts' disks after they come back from a transient disconnection.

For such models, it would be more useful to try to predict long-term availability to select a pool of hosts which can provide the required collective availability over time. This means that out of M selected hosts, at least N of them will be available at any moment, without the need for migrating the service or data. However, long-term prediction may have lower accuracy than short-term prediction, where the system uses the most recent information to predict the immediate future.

One way to perform long-term prediction is to try and detect cyclical patterns in the behavior of hosts. Some works [14] have reported the presence of such kinds of hosts in environments composed of non-dedicated resources, such as volunteer computing systems, enterprise desktop grids and peer-to-peer networks. This paper studies such hosts using trace data of SETI@home [31], and presents possible ways of using them to deploy services in volunteer computing systems with high availability, low redundancy and low number of migrations.

The contributions of this paper are the following:

- We analyze of a large set of trace data extracted from a real system (SETI@home) to identify the presence of different types of hosts (always-on, always-off, cyclic, random) and quantify their presence and contribution to the overall availability of the system.
- We propose a prediction tool (the bit vector) that summarizes the availability information for different times in a week, and assess its quality as a predictor for real hosts in the SETI@home trace.
- We present a method to apply the bit vector-based availability prediction to deploy a service, be it a computational service or a set of data objects (e.g. erasure-coded), and achieve a good level of collective availability using non-dedicated resources. We validate this method using real data, and compare it to random selection and short-term prediction-based methods. We find that the method provides high availability (up to three nines) with low redundancy (a factor of 0.2).

The rest of this document is organized as follows. Section 2 discusses some related work about availability analysis and prediction in distributed systems, with a focus on volunteer computing systems. Section 3 discusses the trace collection method. Section 4 presents some analysis of SETI@home traces, and proves the presence of periodic patterns in host availability. Section 5 presents the bit vector as a tool to predict host availability and assesses its quality as a predictor. Section 6 presents a method which uses the bit vector to deploy services and obtain high collective availability with low redundancy. Section 7 presents the validation of the presented mechanism using the SETI@home traces, and Section 8 concludes.

2. Related work

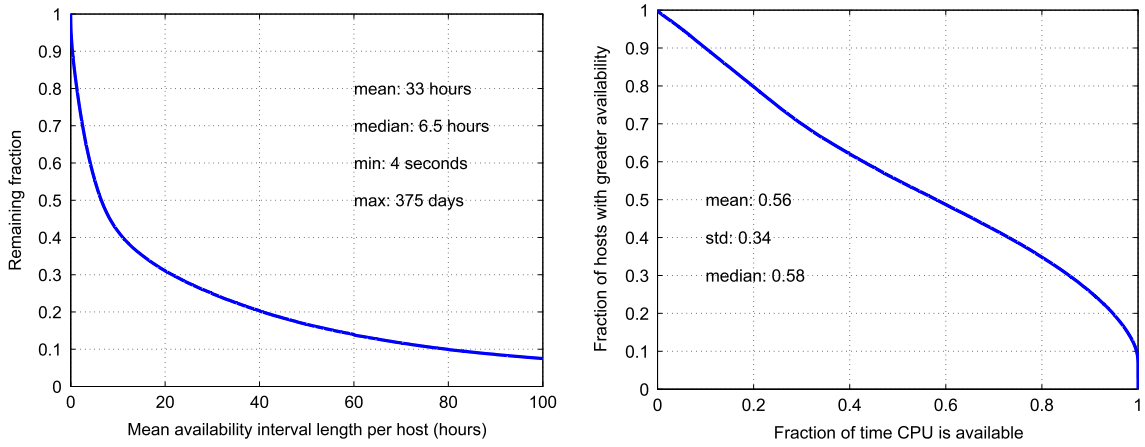
Availability prediction requires obtaining and analyzing trace data of real systems. Many efforts have been devoted to this task in a wide variety of computing environments. Specifically, this work focuses on CPU availability in volunteer computing platforms. CPU availability is different than host availability, which is the focus of previous P2P availability studies [6,24,29,9,14,8]. CPU availability refers to the fact that a host's CPU is available to perform computations in behalf of the distributed systems. This distinction makes sense in cases like volunteer computing systems, where machines are used locally by their owners, and only surplus resources are contributed to the community or to a third party. CPU availability supersedes host availability, as network connectivity is a required but not sufficient condition.

Some studies have analyzed CPU availability by obtaining information about host load [34,8,12,4,13,33]. However, this may not include different causes for CPU unavailability for the desktop, like user interactive activity (mouse or keyboard activity) or OS idiosyncrasies [21] like internal scheduling.

A different way to obtain data about CPU availability in a desktop grid is to submit measurement data that is seen by the workers as a real task. This way, the time assigned to the measurement task, and therefore counted as CPU availability, is the same that it would be to a real task. This method of measurement has been used for some works on different systems. However, the first availability studies of desktop grids only obtained information from a limited number (hundreds) of hosts located in enterprises or universities during a relatively short time period [21].

More recently, some bigger data sets have been published, like the SETI@home trace published in the Failure Trace Archive [22], which contains data about more than 200,000 hosts. This data set has been analyzed for characterizing the behavior of purely random hosts [18]. We use this trace for the analysis and evaluation in this work, and describe it in detail in Section 3.

An important conclusion about availability came from a work by Douceur [14] which studied the distribution of host availability in a few sets of Internet-distributed and enterprise hosts, considering the percentage of time each host is available. He identified this as the graduated mix of two uniform distributions. This could be caused by the existence of two types of hosts, identified by



(a) CPU interval lengths. (b) Fraction of time CPU is available.

Fig. 1. CPU availability.

Douceur as one group of hosts with cyclical behavior, and another of hosts which are left on at all times. This supports our approach of investigating the existence and possible use of hosts with periodical availability patterns.

Prediction is an established technique in the management of computer systems [32,12,16]. Many works have focused on online availability prediction [28,25,3], using up to date data to predict availability in the immediate future. The different approach of capturing long-term availability patterns has also been used in [27], which proposed a summary of the availability of hosts during a week. This is similar to our approach, detailed in the following sections, but it differs in that they used it to select hosts with high expected availability during a period of time to execute a task, whereas we will apply availability prediction for service deployment. Another important difference is that they only tried their mechanisms with a small number of hosts (75) in three labs, while we validate ours with the SETI@home trace, which contains hundreds of thousands of hosts, most of them in home environments.

As stated earlier, service deployment with collective availability is similar to distributed data storage. A possible approach to providing availability guarantees in storage is to compute the required number of replicas (or fragments) of a data object, according to the average availability of hosts in the system [7,19]. One problem of this approach is that it requires knowledge of the average host availability, which may require expensive procedures to be estimated. However, even more important is the fact that these works only consider average availability, as if all hosts' behaviors are identical and independent. According to [5], using average availability without considering correlation, i.e., assuming host independence, highly overestimates the actual availability obtained by a deployment. Similar conclusions about the important contribution of failure correlation to unavailability were found at Google storage systems [10].

In our own previous works, we focused exclusively on either *random* availability patterns [17,18] or *short-term* predictions [3]. In contrast, in this work, we focus on predicting *deterministic or semi-deterministic* patterns of availability over relatively *long time scales* (days versus hours).

3. Trace collection

In order to detect periodical availability patterns, we have used a set of real CPU availability traces from SETI@home, which are part of the Failure Trace Archive [22], publicly available at http://fta.inria.fr. The traces were collected using the BOINC middleware [1] for volunteer computing. BOINC serves as the basis for projects such as SETI@home, EINSTEIN@home, and climateprediction.net. These traces were recorded at the BOINC server for SETI@home, thanks to an instrumented BOINC client. They contain information taken from 226,208 hosts over the Internet, mainly at home locations, which is an important difference with older data sets. The trace period goes from April 1, 2007 to January 1, 2009. In total, the traces capture about 57,800 years of CPU time and 102,416,434 continuous intervals of CPU availability. The traces do not show host availability, but CPU availability, as explained in Section 2. The traces record the exact times when the CPU is available for computation as defined by the BOINC user preferences.

4. Trace analysis

4.1. General characterization

We present a general characterization to quantify the variation in availability among resources and validate our hypothesis that services could be kept available by leveraging hosts with medium availability levels.

Fig. 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 Fig. 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 33 h and 6.5 h respectively. The mean availability lengths are more than 5 times greater than in enterprise environments [23]. About 70% of the mean interval lengths are less than 24 h, indicating the need for fault-tolerance for long-running applications. It is clear that, in order to deploy a service using this set of resources, continuous availability cannot be achieved by relying on a single host's availability intervals, since migrations would be frequently needed.

Fig. 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 Fig. 1(b) means that y fraction of the clients are available more than x of the time.

We observe moderate skew. About 35% 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 possible to model the trend using a least-squares fit. The mean and median

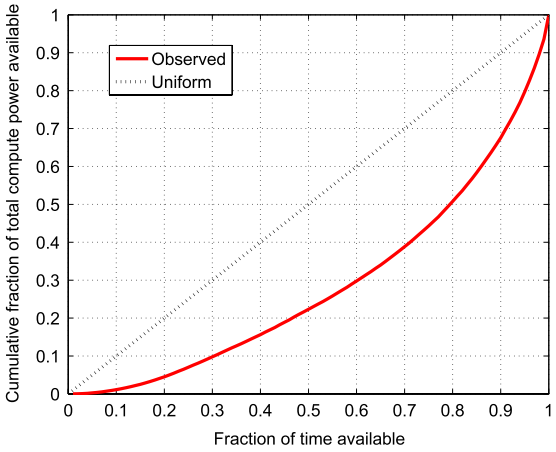


Fig. 2. Cumulative CPU time multiplied by host speed over all hosts.

fractions of time available are 56% and 58% respectively. The mean is more than 1.3 times lower than that observed in enterprise desktop grids [23]. The standard deviation at 34% is quite high, and is about 50% of the mean. We conclude that residential machines are powered-on for less time (by more than a factor of 1.3) than those in enterprise environments, but when powered-on, residential machines are more idle (by more than a factor of 5.25 on average).

These results show that many hosts have a relatively low availability. The next question that arises is how significant is the contribution of these hosts. To investigate this issue, we show in Fig. 2 the fraction of compute power (CPU time multiplied by FPOPS per host)[1] 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 30% of the total CPU time. Nevertheless, hosts with lesser availability contribute significantly. For example, hosts with intermediate availability between 55% and 90% contribute about 45% of the platform's CPU time. Hosts available 55% of the time or less contribute only 25%.

In conclusion, we have seen that there is a significant amount of hosts with relatively short availability intervals (a few hours) and with moderate availability levels. Nevertheless, they contribute a significant fraction of the total CPU availability. This makes it advisable to find ways to use these resources to attain levels of collective availability well above the individual availability of each host. Our approach consists of trying to find patterns in host availability.

4.2. Availability patterns

Our first step to detect patterns in host behavior is to inspect the traces using a visualization software called Pajé [30]. Fig. 3 shows a small random subsample of the trace visualized with Pajé, where each horizontal line represents a host's availability and unavailability periods, i.e., the host is available when a thick line is visible, and unavailable when there is no line. The numbers on top show the days in the trace. The figure shows that there are hosts with clearly periodic patterns (e.g. availability intervals every day during office hours), but at the same time there are others with long availability or unavailability stretches, or with no distinguishable patterns.

Following the method in [20], we represent the availability of each host as a vector of 168 bits (24×7). Each bit represents an hour of the week, and its value is 1 if the host is usually available during that hour, and 0 otherwise. In more detail, we compute the percentage of time the host has been available during each hour in its lifetime (from the beginning of its first availability period in the trace to the end of the last one), and find the average for each hour of the week. Then we convert this value to 1 if it is equal or greater than a certain threshold (binarization threshold), or to 0 if it is lower. For the clustering, we set the threshold to 0.75. By setting it high, we focus only on patterns that are repetitive over a relatively long term.

A measure of the difference between the bit vector and the actual trace is the percentage of false positives. We define it as the amount of time when the host is actually unavailable, but is shown to be available by the bit vector at that time of the week. Since our binarization threshold is 0.75, the maximum percentage of false positives that our bit vectors can show is 25%. This is because the worst case would be a host which is available at most 75% of the time during some hours of the week. Such a host would be considered to be available during those hours, but it would not be actually available in 25% of these cases. An hour for which the host has been available less than 75% of the time would have a 0 on the bit vector. Fig. 4(a) shows that the amount of false positives is much lower than that for most cases, being lower than 1% for about 99% of the hosts.

Similarly, we define the amount of false negatives as the percentage of time when the host was available but the bit vector showed it as unavailable. It is lower than 1% for about 97% of the hosts (see Fig. 4(b)), much lower than the theoretical maximum of 75%. This proves that the bit vector is a good representation of a host's availability.

Our next step is to follow the method in [20], and use clustering to separate hosts with different patterns. The clustering is performed using the k-means algorithm [15]. It works by selecting k random points to use as cluster centroids, and assigning each element to the cluster with the nearest centroid. It does so iteratively, calculating the cluster centroid from the elements in the cluster and recomputing the distances between the elements and the centroids, until no changes occur. It ends when each element is in the cluster with the closest centroid, as calculated from the elements of the cluster.

To determine the similarity between hosts and centroids, we used a Hamming distance metric. Given two binary vectors, this metric measures the fraction of unequal values in each dimension.

The number of clusters k is an input parameter, and the results may vary in each execution because of the random selection of initial centroids. We tried different values of k, from 3 to 30. 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 found that the clustering revealed some dominant clusters that were unmodified by changing k. We chose to show the results with $k = 6$ as, after visual inspection, these are where the patterns were the most pronounced. However, note that we use this clustering only for descriptive purposes.

Our trace contains more hosts and a longer period of time than the one used in [20], but it's of the same system (SETI@home) and the results are highly consistent. The largest clusters are the one with 100% availability (*always-on*) and the one with 0% availability (*always-off*), as reflected in the cluster centroid. Of course this does not mean hosts actually have 100 or 0 availability, nor that their bit vectors say so. It only means that they are close to this behavior. The rest of the clusters comprise in total about 10% of the hosts and

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

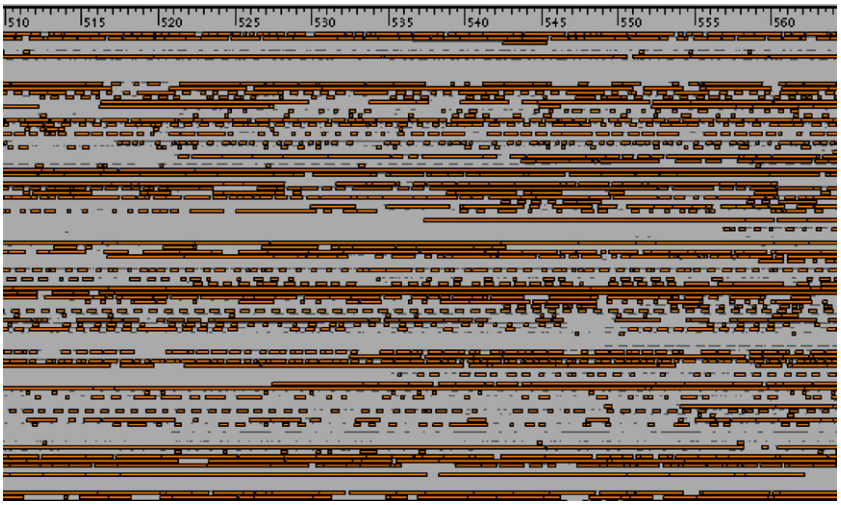
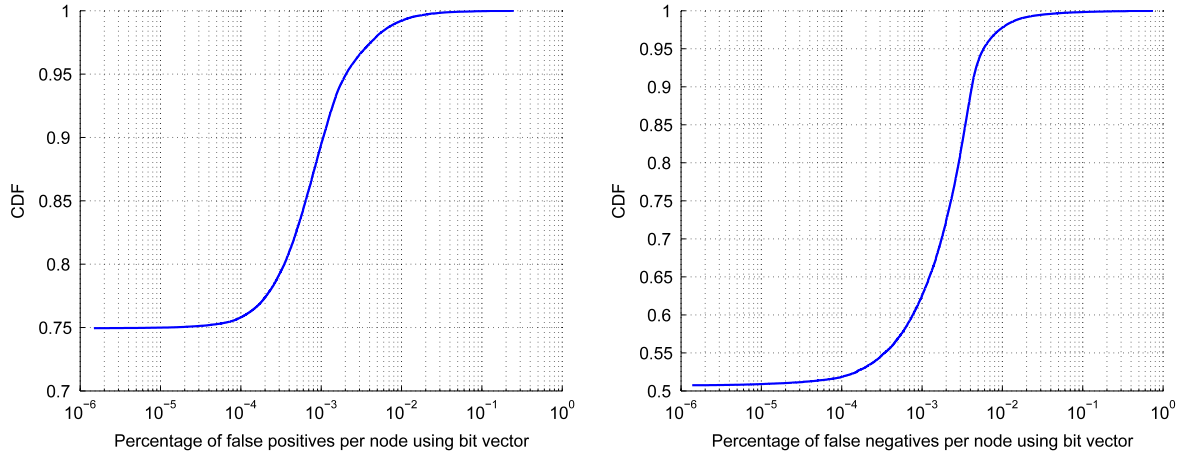


Fig. 3. Trace visualized with Pajé. Each line represents a host's availability, and the numbers show the days in the trace.



(a) CDF of the fraction of false positives for all hosts, in log scale.

(b) CDF of the percentage of false negatives for all hosts, in log scale.

Fig. 4. Difference between the bit vector summary and the actual trace.

show more periodical patterns. Choosing higher values of k further subdivided these cyclical clusters, while leaving the always-on and always-off clusters unchanged. Fig. 5 shows, for each cluster, with $k = 6$, the sum of the bit vectors of all the hosts in the cluster, together with the centroid of the cluster multiplied by the total number of hosts in the cluster. We see that the behavior seen in the cluster centroid closely resembles that seen in the sum of the hosts' bit vectors.

We also take a look at the actual behavior of the hosts in each cluster using Pajé. Fig. 6 shows a small period of time of a random subsample of the hosts in the *always-off* cluster. Each horizontal line represents the availability of a host (a thick line represents an availability period, while no line represents unavailability), and the numbers in the x axis show the days of the trace. Hosts do not have 0 availability, but they have sporadic availability intervals that do not follow identifiable patterns. Fig. 7 shows hosts from the *always-on* cluster, which have long availability intervals separated by short periods of unavailability. Fig. 8 shows the behavior of a subsample of cluster 3, one of the clusters with cyclical availability patterns. Although not all hosts exhibit an apparent periodical pattern, such behaviors are clearly present in the cluster.

5. Online availability prediction

Trace analysis presented in Section 4.2 has shown that the bit vector is a good and accurate summary of the availability of hosts over time for a large fraction of hosts, detecting existing daily and weekly patterns. These patterns are usually repeated over the weeks, with only small punctual variations, or changing over longer periods with seasonal effects. For this reason, we can expect that the patterns detected by the bit vector will also be repeated in the near future. Therefore, we will consider the usability of the bit vector as an availability predictor.

To predict the behavior of a host, its first weeks of life are used as training data to generate a bit vector following the method explained in Section 4.2. This vector can then be used to predict the behavior of the host during the next week: the host is expected to be available during the intervals where the bit vector has a 1, and unavailable during the rest of the week, when the bit vector has a 0.

Metrics for prediction quality. We use two metrics to assess the quality of the predictions made by the bit vector. The first one is the *rate of false positives*: amount of time when the host is unavailable and predicted to be available, divided by the total time the host is predicted to be available. This gives a measure of the reliability of the prediction, i.e., the confidence that a host will be available when the bit vector predicts it to be available. The second metric used is the *rate of false negatives*, which is similarly defined as the amount of time when the host is available and predicted to be unavailable, divided by the total time the host is predicted to be unavailable. This could be considered a measure of the efficiency

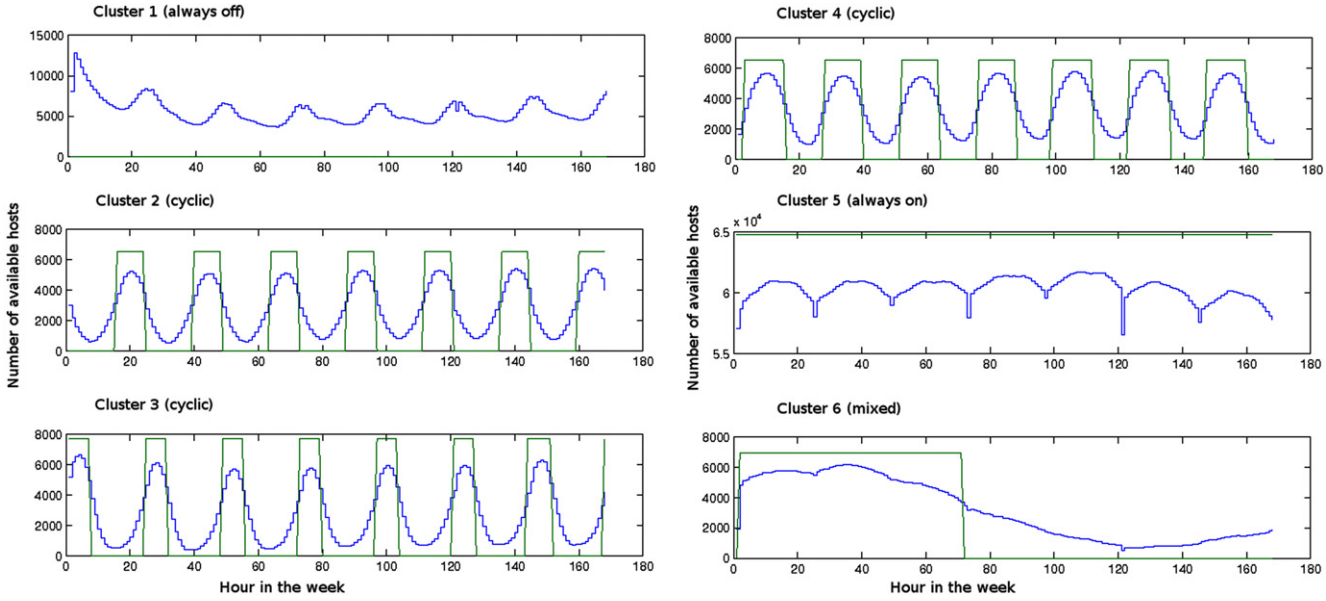


Fig. 5. Sum of the bit vectors of the hosts of each cluster, compared to the cluster centroid multiplied by the number of hosts in the cluster.

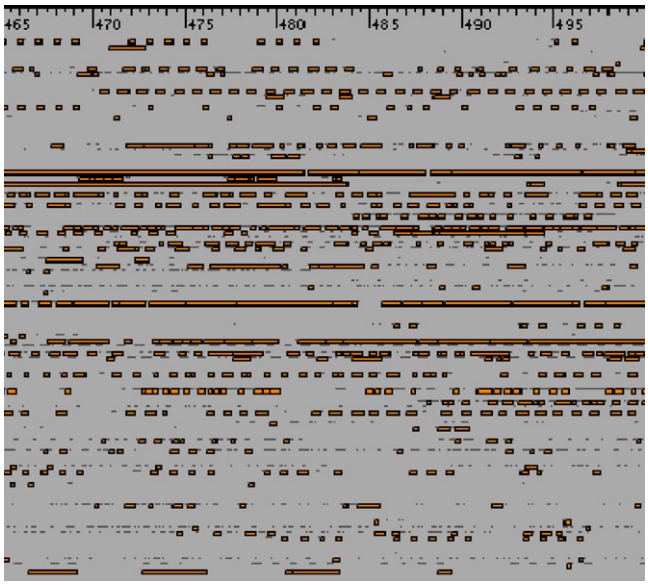


Fig. 6. Trace of a random subsample of the hosts in cluster 1 (always off).

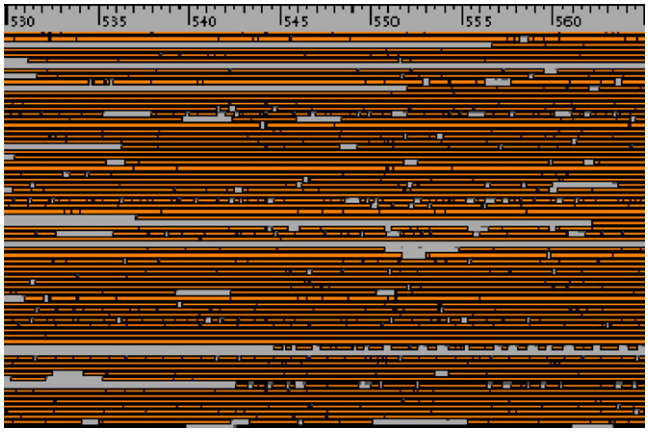


Fig. 7. Trace of a random subsample of the hosts in cluster 5 (always on).

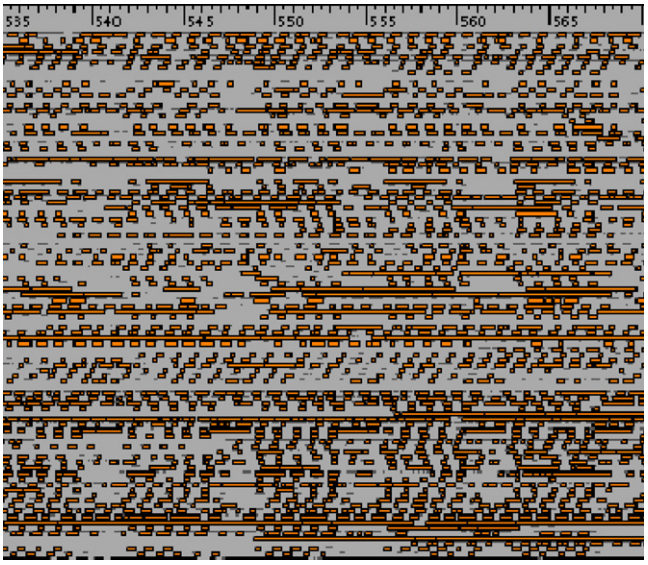


Fig. 8. Trace of a random subsample of the hosts in cluster 3 (cyclic).

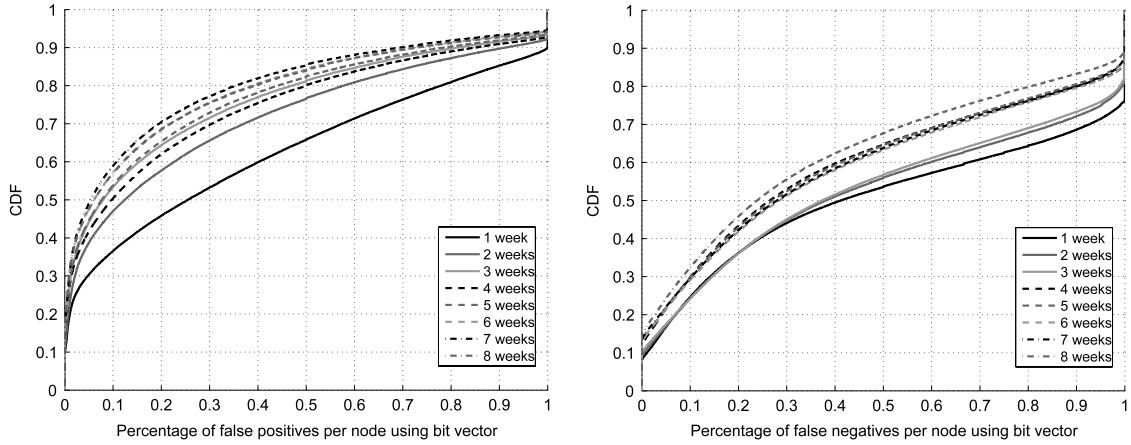
permitted by the predictor. A system relying completely on the availability prediction would not try to use hosts when they are not expected to be available, and therefore their unpredicted time of availability would be wasted.

5.1. Evaluating different parameters for prediction

In this section we will present a variety of prediction parameters, and also show how they affect prediction quality.

Prediction parameter 1: *length of training period*.

Short training periods may not be enough to capture the presence of cyclic patterns. However, requiring very large amounts of data to make the predictions may be a problem in a real system, e.g. because of the space required to store it or because of the time required to obtain it. Fig. 9(a) and (b) show the amount of false positives and false negatives obtained when creating the bit vector with different amounts of information, from 1 week to 8 weeks. The threshold used is 0.75, and the interval length used is one hour. These plots do not show the hosts with 0 availability for the



(a) CDF of the false positives.

(b) CDF of the false negatives.

Fig. 9. Prediction error obtained when using a bit vector (generated with different number of weeks of information and threshold 0.75) to predict the behavior of the next week.

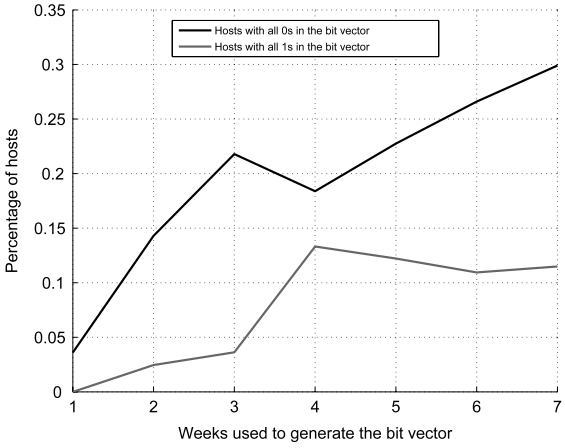


Fig. 10. Amount of hosts with availability predicted as 0% and 100% with different number of weeks.

false positives. Although these have naturally no false positives, the prediction is useless, because a prediction mechanism would consider them unavailable and therefore unusable. Fig. 10 shows the amount of hosts with availability predicted as 0 by the bit vector for different number of weeks. Similarly, we have excluded hosts with 100% availability predicted by the bit vector from the false negatives plot.

Between 9% and 18% of hosts, approximately, have no false positives at all, depending on the training period length (see Fig. 9(a)). 50% of the hosts, or more, have less than 10% false positives when using a training period longer than 2 weeks. However, not all hosts are this predictable, and there is even an amount almost as large as 10% in the worst case with 100% error (false positives). There is a large difference between using one week to generate the bit vector and using more, but when using more than 2 the improvements obtained by using one more week are moderate.

Regarding false negatives (see Fig. 9(b)), however, the biggest difference is between using 3 and 4 weeks as a training period. Using 4 or more weeks, 50% of the hosts have less than 30% of false negatives, but even in the best case, there are more than 10% of the hosts with 100% false negatives, and the error is higher than when considering false positives. This is consequent with the fact that our binarization threshold is relatively high (0.75), therefore prioritizing reliability over efficiency. Another consequence of the conservativeness of our method is the fact that a high amount of hosts (more than 20% for many configurations) have no predicted availability (see Fig. 10).

Prediction parameter 2: *bit vector granularity*.

We define the granularity of the bit vector as the length of the intervals in which the week is divided. A fine granularity may give more accurate predictions, but it also requires more data in the bit vector. This may be an issue if the data storage space is limited, or if the bit vectors are to be sent through a network. We have considered that one hour is a coarse enough granularity, so we have not tried intervals longer than that. On the other hand, we have considered that intervals as short as one minute would generate a huge amount of data that would make them almost intractable. The intervals lengths we have tested are 60, 30 and 15 min. Figs. 11(a), (b) and 12 show, however, that there is almost no difference between these granularities.

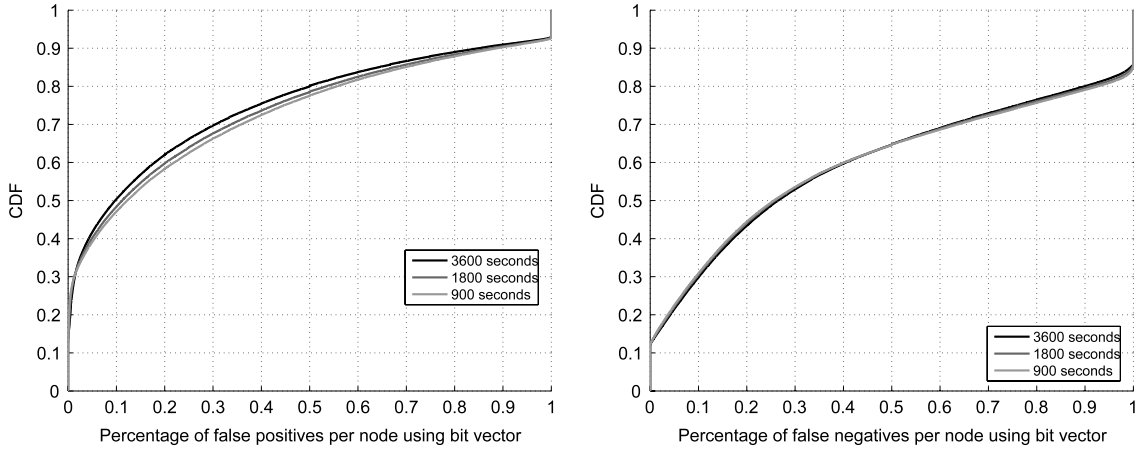
Prediction parameter 3: *threshold for binarization*.

Our purpose is to detect patterns in the behavior of hosts, by finding the intervals when the host is almost always available, and those when it is almost never available. In this ideal case, all values would be near 0 or near 1 before binarization, so the threshold would not be important. However, because in the real trace there are many values in the middle ground, we need to set the threshold carefully so that we do not predict availability for intervals when a host is only sporadically available, but we do predict availability when it is due. For this purpose, we have tried different thresholds, ranging from 0.55 to 0.95.

As expected, the higher thresholds cause a lower rate of false positives, but a higher rate of false negatives (Figs. 13 and 14). However, there is a larger difference between the thresholds 0.75 and 0.85. This may mean that there are a large number of hosts with an average availability of near 80% during some hours. These intervals are considered as available when using thresholds equal or lower than 0.75, but unavailable when using a threshold of 0.85. Predicting such intervals as available or unavailable makes a great difference. Therefore, a threshold of 0.85 will give us pessimistic predictions, where a 1 in a bit vector positions means almost total guarantee that the host will be available. A threshold of 0.75, on the other hand, can be used to make more flexible predictions. We will take this into account in Section 7.

Prediction parameter 4: *length of prediction interval into the future*.

Until now, we have only tried to predict the behavior of hosts during one week using the availability data of immediately previous weeks. In a real system, the bit vector could be generated every week with the latest data and then used to predict the next week. However, if the results of predicting the next N weeks are



(a) CDF of the false positives. (b) CDF of the false negatives.

Fig. 11. Prediction error obtained when using a bit vector (generated with 4 weeks of information and threshold 0.75) to predict the behavior of the next week, using different binarization interval length (in seconds).

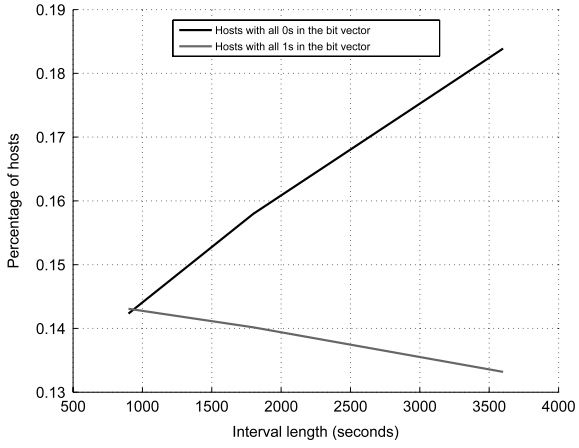


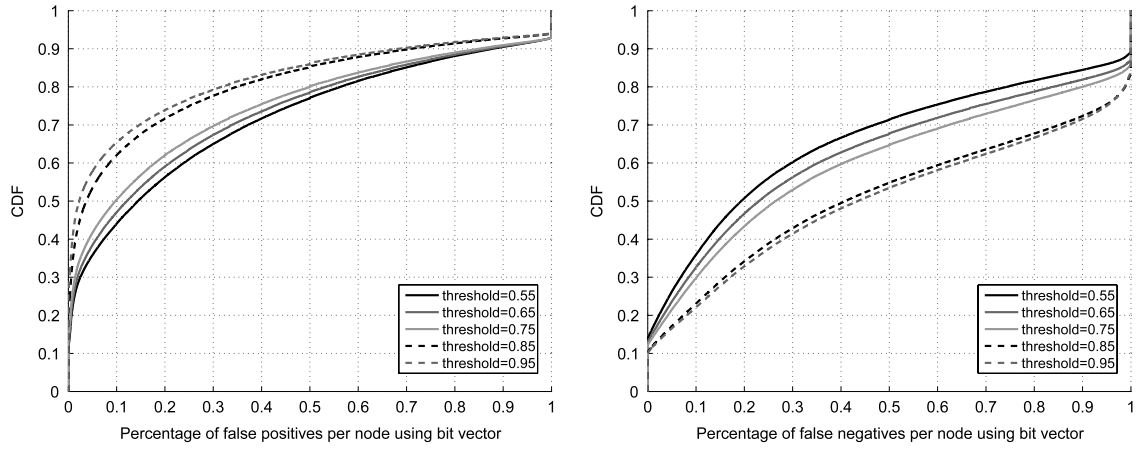
Fig. 12. Amount of hosts with availability predicted as 0% and 100% with different binarization interval length (in seconds).

not much worse than predicting the next $N - 1$, we could say that the prediction does not degrade excessively over time. If this is the case, a system may not need to generate and apply a new bit vector every week.

Fig. 15 shows how the quality of predictions degrades when using the bit vector generated with the first 4 weeks of life of each host to predict its behavior during the next 1–8 weeks, and during the rest of the trace. The biggest difference in false positives is between predicting 1 and 2 weeks, while the difference when predicting more weeks is smaller. It must be noted that, while the amount of hosts with low prediction error decreases over time, the amount of hosts with high prediction error also decreases. This is probably because these are hosts with a highly random behavior, so when taking more time (more samples) the proportion of extreme values is lower. The degradation may be explained by the effect of seasonal variations, e.g. holidays. This may cause the big difference seen when trying to predict the whole trace, since in such a long period seasonal effects may be more present.

6. Availability-aware service deployment

In the previous section we have presented the bit vector as an availability predictor for individual nodes and assessed the quality of its predictions under different configurations. The next step in our work is to use this predictor to provide collective availability to a service while minimizing redundancy. We will do this through permanently deploying the service in a set of hosts with (negatively) correlated availability patterns. In other words, if a service needs N available instances to be considered available, it will be deployed in $M \geq N$ hosts which will be executing the service whenever they are available (the definition of availability is



(a) CDF of the false positives. (b) CDF of the false negatives.

Fig. 13. Prediction error obtained when using a bit vector (generated with 4 weeks of information) to predict the behavior of the next week, using different binarization thresholds.

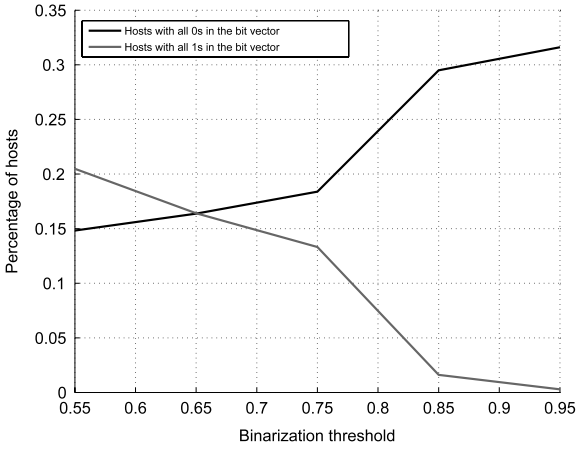


Fig. 14. Amount of hosts with availability predicted as 0% and 100% with different binarization thresholds.

when the CPU is available for computation, as defined by the user preferences). The service is considered to be available at a given moment if at least N of the total M hosts are available.

An alternative application would be the storage of files in the form of erasure codes [11], which requires retrieving N out of M existing fragments to restore the file. However, in this case the definition of availability would be different (host availability would be probably enough, instead of CPU availability).

The remainder of this section will explain how host selection is performed in order to achieve a certain level of collective availability using the bit vectors, and how this deployment could be improved by considering the different predictability levels of each host, incorporating an additional redundancy factor or detecting permanent host departures.

Host selection for collective availability. A process, called the service deployer, uses the bit vector to decide which hosts to use for the deployment of the service, as it captures the periodical availability patterns of each host. This information is obtained through an independent availability monitor. Each host is expected to be available during the intervals for which it has a 1 in its bit vector. Therefore, the sum of the bit vectors of a set of hosts shows how many of these hosts are expected to be available at each interval.

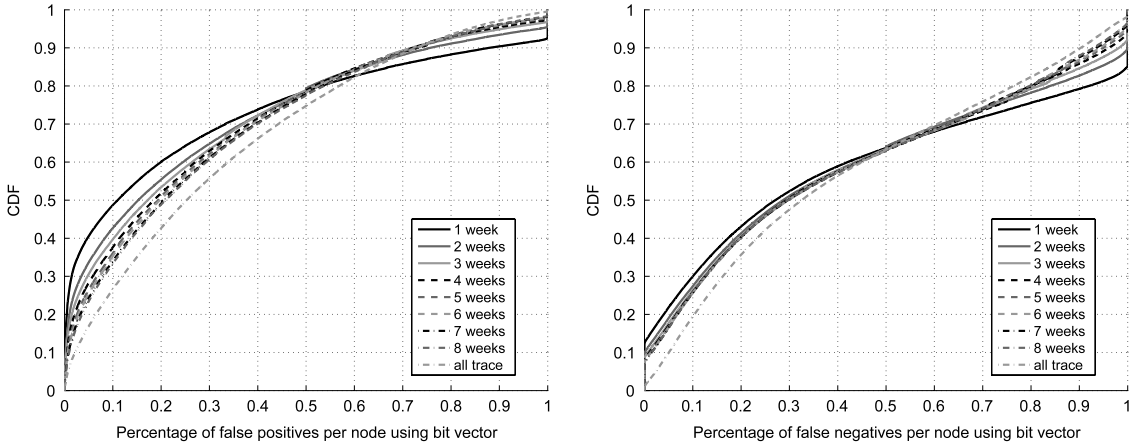
The service deployer picks hosts randomly, and keeps only those that bring the set closer to the target, i.e., a sum vector with a value equal or greater than N in each position. It picks hosts until the sum vector of the selected hosts has reached the target. Then it starts a second round where it checks if each of the hosts in the selection is contributing to reach the target. It removes hosts that can be taken out of the selection with the sum vector of the remaining set still reaching the target.

This is a very simple approach, but it has the advantage that its simplicity allows it to be used even in decentralized environments where the deployer has no complete knowledge of the hosts in the system. It clearly gives preference to availability over efficiency, since we will tolerate the assignment of more resources than needed (having values greater than N in some positions). A more complex approach could involve trying to find the hosts that optimize the sum vector, to have at each position a value as close to N as possible.

Predictability. The previous analysis has shown that the bit vector is a good predictor for some of the hosts, but it is not so good for some others. Therefore, host pre-selection could improve the overall performance of the deployment mechanism. This means restricting the hosts we consider for deployment, picked before looking at their bit vectors. For example, it would be useful to prioritize the use of more predictable hosts. The most obvious metric for predictability would be the accuracy of a prediction done using the same model. In our case, if we have a trace of W weeks to generate the bit vector, we could generate a bit vector using the first $W - 1$ weeks of the trace and compute the false positives obtained when comparing it to the last week of the trace. This could give us a measure of how predictable the behavior of the host is. Other metrics we can use to prioritize host selection are average availability, or the clustering done in Section 4. We will test the effect of such pre-selection criteria in Section 7.

Redundancy factor. Another tool that can be used to counter the possible lack of accuracy in the prediction for some of the hosts is a redundancy factor. Instead of selecting hosts until reaching a target of N in all the positions of the bit vector, the deployer can artificially add more redundancy by using a redundancy factor $R \geq 1$, and setting the target as $R \times N$. This way the deployment can tolerate the unexpected availability of some of the hosts in the set.

Detection of host's system departure. Finally, the proposed deployment mechanism should be complemented with a method to detect permanent departures of hosts. When this happens, one or more new hosts can be selected by the deployer using the same mechanism. This is necessary because our availability prediction method does not predict permanent departures, so only relying on it cannot guarantee durability. Some of the selected hosts may leave the system permanently over time and the service may become unavailable because of this. A simple method to provide durability is to re-evaluate the deployment every week



(a) CDF of the false positives.

(b) CDF of the false negatives.

Fig. 15. Prediction error obtained when using a bit vector (generated with 4 weeks of information and threshold 0.75) to predict the behavior of the following weeks.

using the most recent availability data to generate a bit vector for each host. Hosts that have permanently left will end up having a bit vector of all zeros, and therefore they will be removed from the deployment. This method would also prevent performance degradation caused by seasonal effects, i.e., variations in the behavior of hosts. Alternatively, hosts that are offline for more than a given period (e.g. one week) could be removed instantly from any deployment.

7. Evaluation via trace-driven simulation

In order to validate the resource management mechanism presented in the previous section, we have used the SETI@home traces to simulate service deployment on volunteer resources and evaluate the availability and efficiency obtained by our methods. We have compared them to other methods which do not use availability prediction, and found that our method outperforms them in both provided availability and resource usage efficiency.

In order to evaluate the performance of our mechanism under different conditions, we have tested it changing parameters like redundancy factor, required number of available hosts, deployment length and pre-selection policies.

7.1. Test environment

Simulation method. We validate the presented resource management mechanism by simulation using the SETI@home traces. Given a certain point in time to be used as the beginning of the experiment, we calculate the bit vector of each host using the previous 4 weeks of traces, as Section 5 showed it to be a long enough training period. The deployer uses these bit vectors to select a set of hosts to deploy the service as previously explained. Then we look at the behavior of the hosts after the starting point and see if the service achieves the required collective availability during the next week.

We repeat the simulation with three different starting dates, one corresponding approximately to one fourth of the trace, another near half of the trace, and a last one near three fourths of the trace's length. The three starting dates are at Monday 00:00, so we can make predictions for a whole week. Theoretically, there should be no difference when deploying a service in any other moment of the week. Since there is a random component in our deployment mechanism, we repeated the simulation 1000 times with each starting point. For each configuration tested, we take the results obtained with the three different starting points (3000 simulations in total) and average the results.

An important parameter of the simulations is the desired number of hosts N. We have set 50 as the default value, although we have tried different values to see the effects of this parameter on the performance of the deployment method, as shown in Section 7.2.

Reliability metrics. The main metric we use to evaluate the reliability of the deployment is the *achieved availability*, which is the percentage of time the number of available hosts is equal or higher than required. An additional metric we measure is the number of required hours of host availability that have not been provided, or *missing host-hours*. It is equivalent to the length of an interval where there are less than N available hosts, multiplied by the difference between the number of required hosts and the available hosts. This is put in perspective by dividing it by the number of required host-hours of availability. We also measure the *success rate*, which is the fraction of simulations where the service is available during the whole week.

Efficiency metrics. We also evaluate the efficiency of our approach by measuring the number of hours of host availability that are above the required. Dividing this by the number of required host-hours of availability gives a measure of the *redundancy* provided by the system. Note that this only measures those periods when the service has more available hosts than needed. The missing host-hours, on the other hand, measure only the periods where there are less available hosts than needed. Therefore, these two metrics are independent and do not compensate each other, as it would happen if we only computed the total number of host-hours attained for the service and compare it to the required number of host-hours ($N \times h$, being N the required number of available hosts and h the length in hours of the considered period).

Another more straightforward metric of the performance of the deployment method is the *number of hosts* that it selects for deployment. Although this is not necessarily related to the provided redundancy (e.g. selecting a number of hosts with low availability can provide lower redundancy than selecting fewer hosts with higher availability), each host selected has a cost in the form of transmitting data to that host. Therefore, selecting a low number of hosts is also a sign of efficiency.

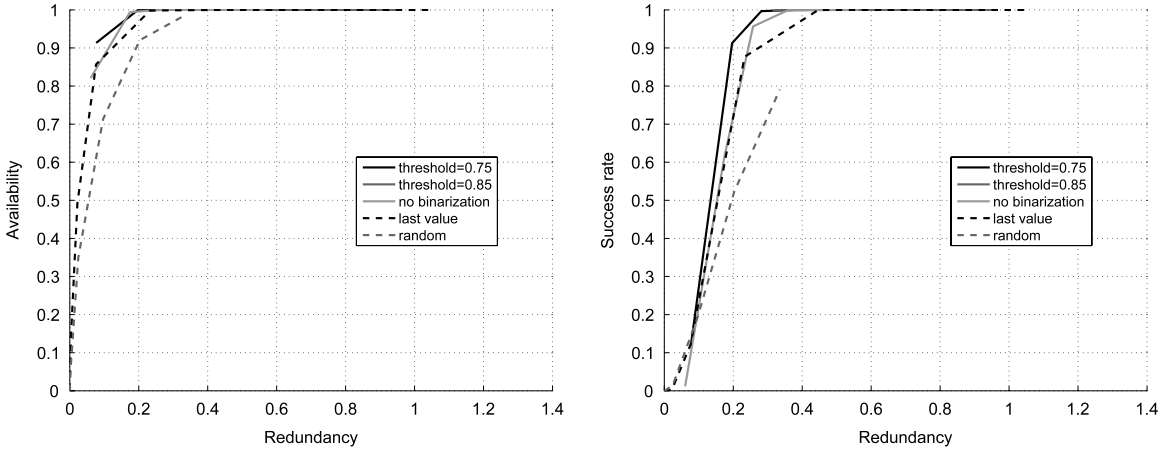
Different predictive methods compared. We have selected two different threshold values to generate the bit vectors used in the simulations: 0.75 and 0.85. The reason is that the results for these thresholds show the biggest difference, as seen in Section 5. A third option we have tried is to use the availability vectors without the binarization step. This gives a vector of 168 positions, showing the average availability of the host for each hour of the week. We sum these vectors directly, and stop the deployment when the sum vector is equal or greater than N for all positions. Finally, we have also tried two more methods to compare against our prediction-based deployment. These are *last value*, which just selects hosts that are available in the moment of deployment, and *random*, which consists of simply picking a number of hosts randomly.

7.2. Evaluation results

Evaluation of redundancy factor. The first step to compare the performance of these five deployment methods is to determine the right *redundancy factor* required for each of them. Note that this *redundancy factor* is an input parameter, not to be confused with the *obtained redundancy* previously defined as an efficiency metric. In the methods based on bit vectors, it is the R that we use to compute the required number $R \times N$ in each position of the sum vector, as explained in Section 6. For *last-value* and *random*, $R \times N$ is directly the number of hosts that we select. Therefore, these replication factors are not comparable and must be set separately for each method.

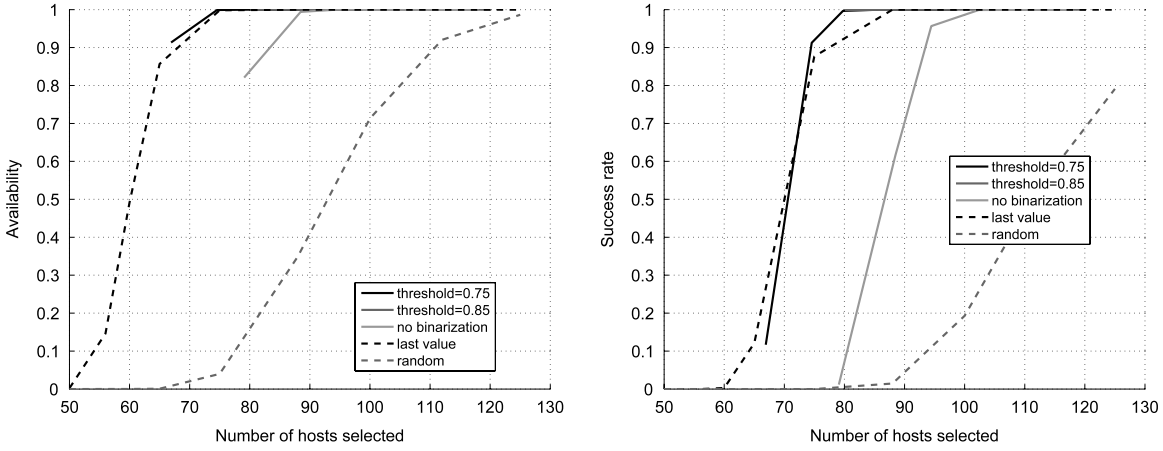
We have tried each method with different redundancy factors, and taken the availability and redundancy results obtained with each configuration. Figs. 16 and 17 show the results obtained by each method with equivalent redundancy levels, in percentage of surplus host-hours and in number of selected hosts. The plots do not show the redundancy factor which was used to obtain each result, since, as we mentioned, redundancy factors for different methods are not directly comparable.

Bit vector methods perform consistently better than the rest. Using $threshold = 0.85$ gives 100% availability and success rate in all cases, although the redundancy is higher than required to obtain the same results with the $threshold = 0.75$ (because the redundancy is higher, the line for $threshold = 0.85$ starts more to the right of the figure than the rest and is hardly visible). Using the vectors without binarization gives relatively good results in redundancy, but requires picking a larger number of hosts. *Last value* performs remarkably well, probably because of the starting dates selected: 00.00 at Mondays may be a time when mostly highly available hosts tend to be connected. This hypothesis will be later evaluated. Finally, *random* performs poorly, requiring the



(a) Availability versus redundancy. (b) Success rate versus redundancy.

Fig. 16. Reliability obtained for different levels of redundancy.



(a) Availability versus number of hosts selected. (b) Success rate versus number of hosts selected.

Fig. 17. Reliability obtained for different number of hosts selected.

highest number of hosts and having a higher redundancy for comparable availability.

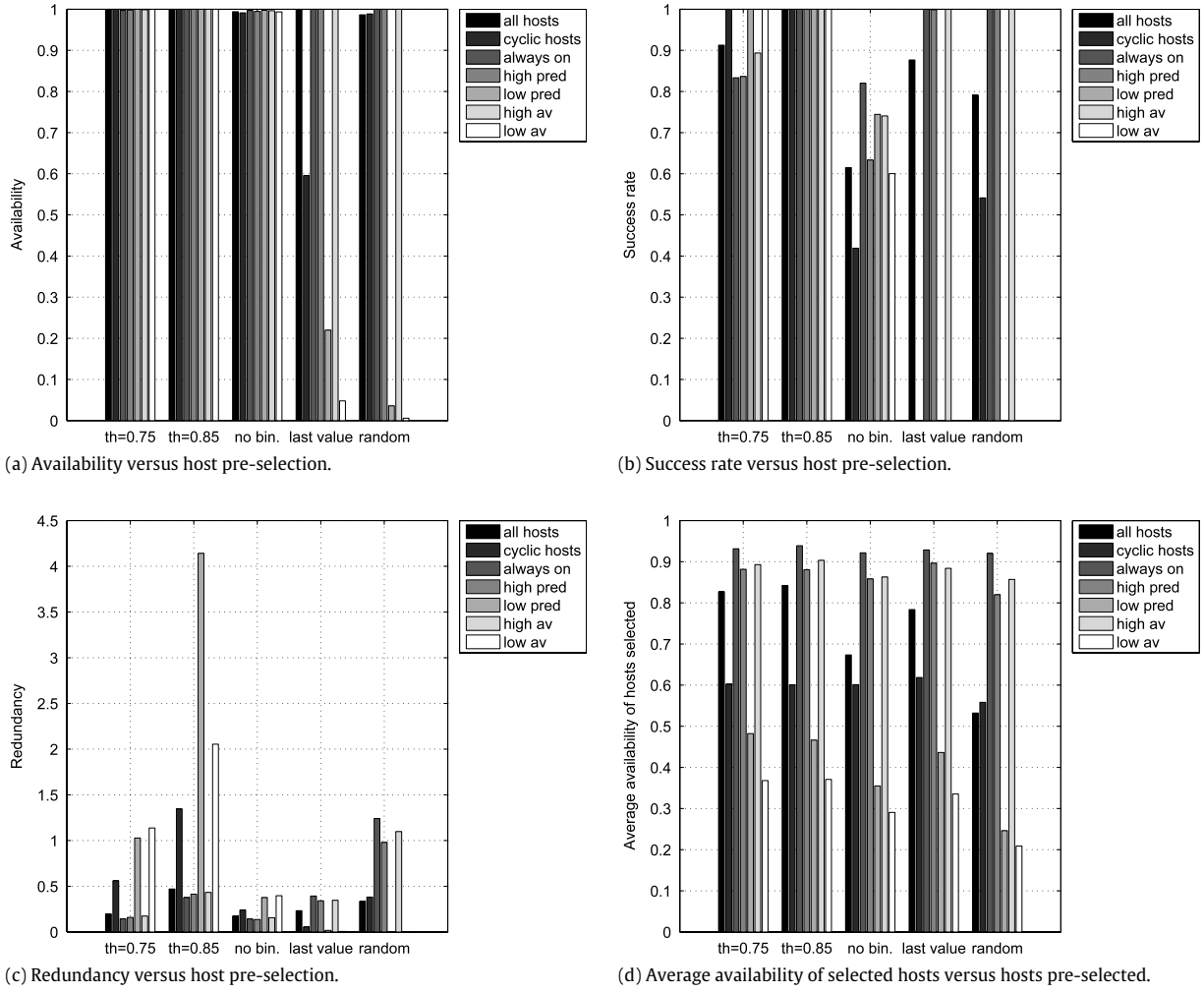
Looking at these results, we select a default value for R for each method. This value should give good results for the method, but it should not be so good (nor so bad) that it masks the effects of changing other parameters. With this in mind, we select $R = 1.1$ for the methods based on the bit vectors, $R = 1.5$ for *last value* and $R = 2.5$ for *random*.

Evaluation using different host pre-selection policies. As said before, host pre-selection may have a great impact on the performance of service deployment. Moreover, it would also be interesting to check if our prediction methods could leverage hosts with mid/low availability. Specifically, hosts with cyclic patterns should be useful for our bit vector-based methods. Therefore, we tested the five previously explained deployment methods with different subsets of the hosts that appear in the SETI@home trace. One division is using the predictability, as explained in Section 6, to divide the hosts in two subsets: those with high predictability and those with low predictability. In order to perform this division, we sort the complete set of hosts by predictability and divide the list in half. Similarly, we have divided hosts in subsets of high and low availability, using the availability reflected in their bit vectors. Finally, we have used the results of the clustering in Section 4 to separate hosts in always-on and cyclic hosts.

Fig. 18(a) shows that the bit vector-based methods perform well with any subset of hosts, offering an availability very close to 100%, while *last value* and *random* are very sensitive to host pre-selection and perform very poorly in some cases. The use of bit vectors allows a lower redundancy when using *threshold* $= 0.75$, usually lower than with the non-predictive methods in the cases where availability values are similar (with the exception of the *random* method with the *cyclic hosts* subset, which offers slightly lower availability than *threshold* $= 0.75$ with a lower redundancy). The use of *threshold* $= 0.85$ provides 100% availability and success rate in all cases, but at the cost of a higher redundancy, specially when using hosts with lower availability or predictability.

This is caused by the fact that the number of hosts selected is automatically decided based on their availability when using the bit vectors. *Last value* and *random*, on the other hand, pick a fixed number of hosts, and therefore it is natural that their performance heavily depends on the qualities of the pre-selected hosts. While these non-predictive methods could be benefited by a careful host pre-selection, the bit vector-based methods do not need to perform this step. Moreover, they can obtain good results when using sets of hosts with widely different characteristics, i.e., hosts with either high or low availability or predictability.

Surprisingly, however, with *threshold* $= 0.75$ the success rate is lower for the '*always on*' cluster and for hosts with high predictability and high availability (see Fig. 18(b)). This is nevertheless consistent with the fact that these are the configurations where a lower redundancy is obtained (see Fig. 18(c)) and where a lower number of hosts is selected (see



(a) Availability versus host pre-selection.

(b) Success rate versus host pre-selection.

(c) Redundancy versus host pre-selection.

(d) Average availability of selected hosts versus hosts pre-selected.

Fig. 18. Reliability and efficiency metrics for different host pre-selection policies.

Fig. 19). That is because all these pre-selections favor hosts with higher availability (as seen in the bars of Fig. 18(d) that belong to the random configuration, which only takes a random subset of hosts and is therefore representative of each host set). This allows the bit vector-based methods to make a deployment with low redundancy, as many hosts have 1 in all or most of the positions of their bit vectors, and therefore a number of hosts close to $R \times N$ causes the sum vector to have $R \times N$ or more in all positions. On the contrary, using cyclic hosts or hosts with low availability requires more complex combinations of hosts, and causes a higher redundancy.

On another note, Fig. 18(d) confirms the hypothesis that *last value* tends to select hosts with higher availability, as reflected in the difference between the average availability of hosts selected by *last value* and *random*, especially when no pre-selection is used.

Evaluation for different numbers of required available hosts. Fig. 20(a) shows availability obtained for different numbers of required available hosts (N). The availability is high (over 0.9) in every case, but it is better with higher values of N (over 0.999 for all methods with $N = 1000$). However, the bit vector-based methods have very good results even with $N = 10$ (availability higher than 0.999 with $threshold = 0.85$, and higher than 0.98 with $threshold = 0.75$), while the non-predictive methods do not perform so well.

The success rate, however, is much lower with smaller values of N (see Fig. 20(b)), and in general all methods perform better with higher values of N, having less missing capacity (see Fig. 20(c)),

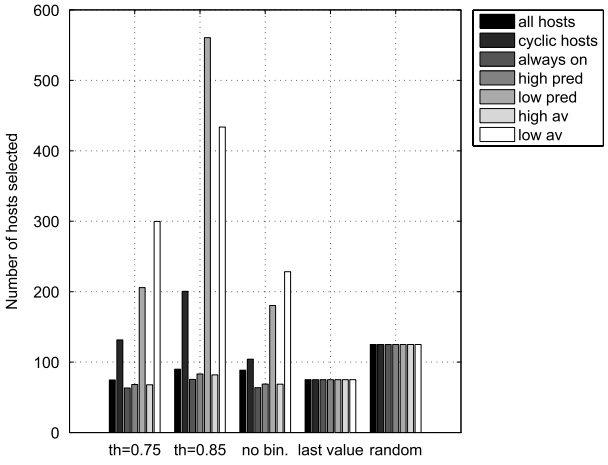
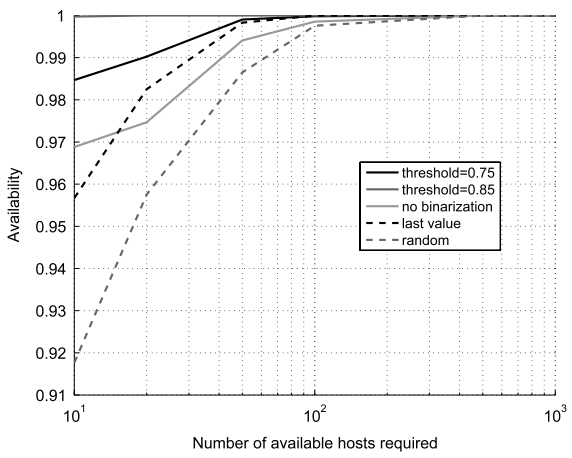
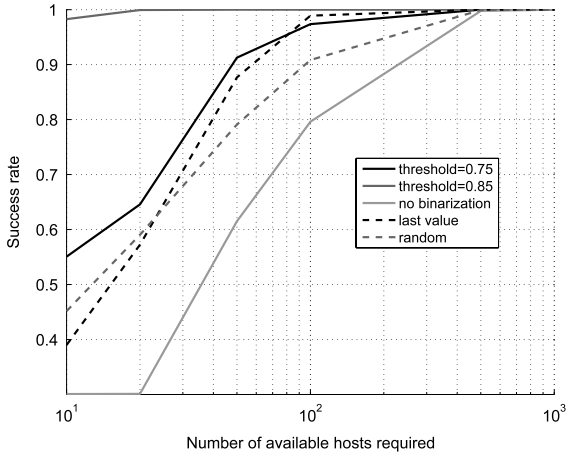


Fig. 19. Number of hosts selected versus host pre-selection.

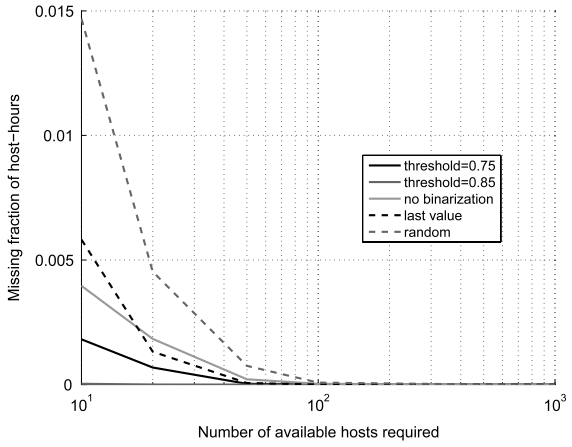
and also less redundancy (see Fig. 20(d)) in the case of vector-based methods. This may be caused by random perturbations in the behavior of hosts, that affect more the smaller deployments as stated by the Law of Large Numbers. To counter this effect, a higher redundancy factor should be set for small deployments in a real system. In any case, we see once again that $threshold = 0.75$ provides either a higher availability than non-predictive



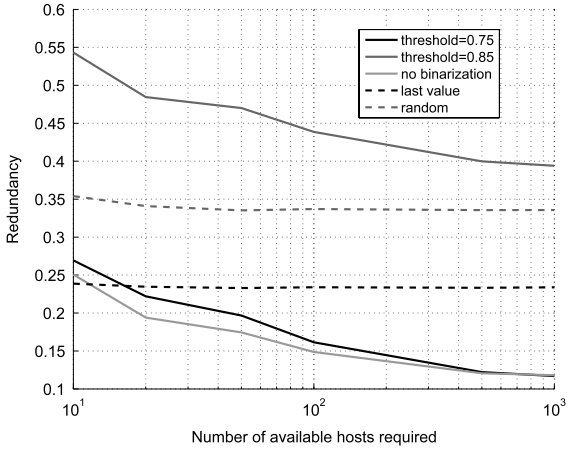
(a) Availability obtained for different numbers of required hosts.



(b) Success rate for different numbers of required hosts.



(c) Fraction of missing host-hours for different numbers of required hosts.



(d) Fraction of surplus host-hours for different numbers of required hosts.

Fig. 20. Reliability and efficiency metrics for different numbers of required hosts.

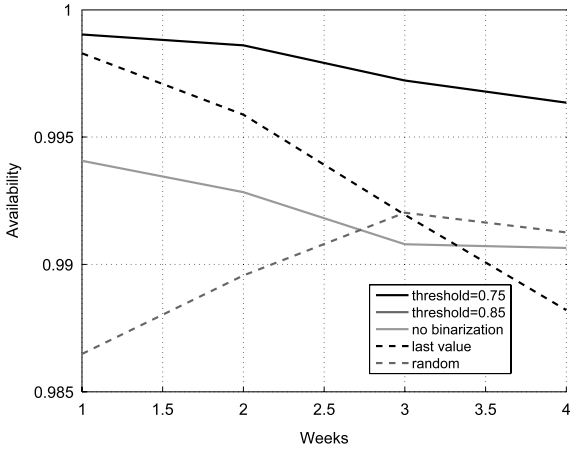
methods with similar redundancy or similar availability with lower redundancy, being in most cases better in both aspects.

Evaluation for different lengths of predicted intervals. The predictions made by the bit vector degrade over time, as seen in Section 6. It would be therefore advisable to re-evaluate a deployment periodically using the most recent availability data to generate a new bit vector for each host. To see how frequent such a process should be, we have tested deployments over longer periods, up to 4 weeks.

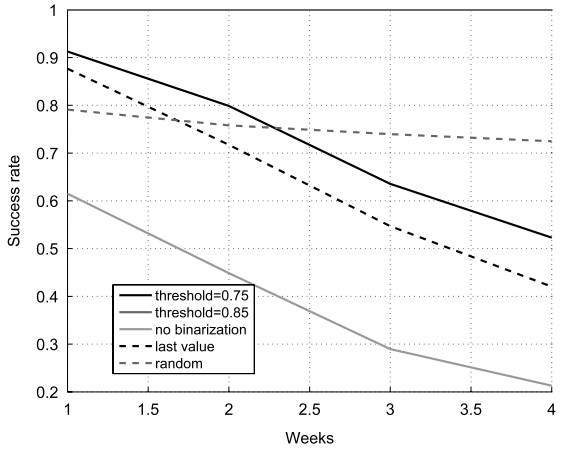
Average availability (see Fig. 21(a)) decreases very slightly over the weeks with the vector-based methods (except for *threshold* = 0.85, which maintains a total availability), while it does degrade more for the *last value* method, falling below 99% after 4 weeks. Redundancy is decreased accordingly (see Fig. 21(d)), while remaining higher for non-predictive methods than for *threshold* = 0.75 or *no binarization*. Success rate, however, suffers a deeper degradation (see Fig. 21(b)). Considering the whole picture, we must note that success rate is the number of times when the service is available during the whole simulation (of a total of 3000 executions for each configuration) and therefore a single second of failure during the simulation can drastically affect its value. Low success rates may therefore be simply caused by the higher failure probability over a higher period of time, more than the degradation of predictions. This is consequent with the fact that the amount of missing host-hours is extremely low in all cases (Fig. 21(c)). Surprisingly, though, *random* suffers the smallest degradation, with only a slight decrease in success rate, and actually increasing its availability, although very slightly. The cause may be the high redundancy of *random* deployment, only second to *threshold* = 0.85 (see Fig. 21(d)).

Comparison with dynamic deployment and short-term prediction methods. The purpose of our deployment method is to leverage knowledge of cyclical patterns and general long-term behavior of hosts to provide high levels of collective availability while minimizing redundancy and communication costs (which is partially determined by the number of migrations). Until now, we have considered that the system only performs the initial deployment, and works with the availability that it provides. However, it could use temporal assignments to increase availability: when the number of hosts becomes $n < N$, the deployer could select $N - n$ hosts that are available at the moment and temporarily deploy the service in them. When the number of hosts available in the permanent deployment set is again equal or greater than N, the hosts that were used for temporary deployment stop executing the service. Note that in some cases, if the unavailability period is short, it may not be efficient, or even useful, to perform a temporary deployment. However, we will consider that the deployer always performs temporary deployments when required, and estimate the total required number of deployments (initial + temporary).

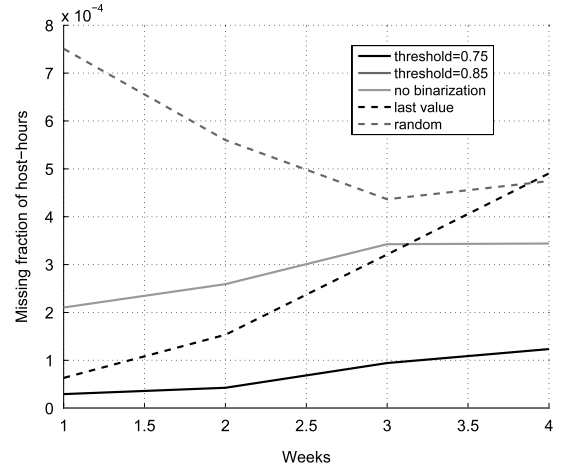
We perform an optimistic estimation. First we count the number of violation intervals, i.e., the time between the point when a host disconnects and the point when that or another host comes back and returns the total number of available hosts to the previous level. There can be different levels of embedded violation



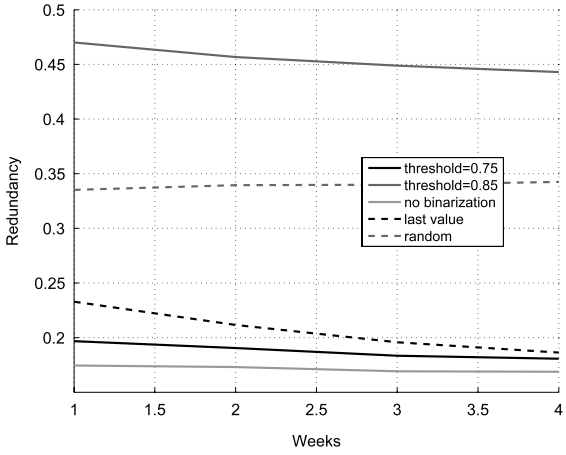
(a) Availability obtained during different number of weeks after service deployment.



(b) Success rate in different number of weeks after service deployment.



(c) Fraction of missing host-hours in different number of weeks after service deployment.



(d) Fraction of surplus host-hours in different number of weeks after service deployment.

Fig. 21. Reliability and efficiency metrics measured in different number of weeks after service deployment.

intervals, e.g. a host disconnects starting an interval with $N - 1$ hosts; then another one leaves and starts an *embedded* interval with $N - 2$ hosts; one host becomes available again and ends the period of $N - 2$ hosts; finally, another host arrives and ends the interval with $N - 1$ hosts. We consider that each of these intervals requires looking for a host outside of the set assigned to the service and perform a temporary assignment, which will end when the interval, as originally computed, ends. This estimation is optimistic because it does not consider the possibility that the host selected for temporary deployment may disconnect before the interval ends. However, since unavailability intervals are often short in our tests, we consider that this is a reasonable assumption.

We perform this estimation with all the previously used methods, and compare it to a simple dynamic deployment approach. It consists of randomly choosing a set of N hosts to deploy the service. It does not use permanent assignments, and therefore whenever a host of the set becomes unavailable, a migration is required. When that happens, another host is chosen randomly among those available at the moment. Its availability could be improved by adding a replication factor. However, for simplicity we will consider that it only deploys the service in N hosts and assume that even in this situation it can provide a sufficiently good availability. Since adding replication would increase the number of migrations, we will compare the bit vector-based methods to the best possible results of the dynamic deployment method in number of deployments.

We also compared our approach to [3], where a short-term prediction is used to achieve collective availability for a service during a given prediction interval, with a typical length of a few hours. The obtained turn-over rate, i.e., the amount of hosts that need to be migrated from one prediction interval to the next one to keep guaranteeing the required availability, is about 2% of the total assignment in average, when using high predictability hosts, with a prediction interval length of 4 h. Note that this total assignment may include a certain redundancy factor R (from 0, i.e., no redundancy, to 0.5 of the required number of hosts) to increase the probability of providing the required availability. This means that the total number of deployments (counting both initial deployment and migrations), with zero redundancy, normalized to the required number of available hosts, would be $1 + t \times i$, t being the turn-over rate and i the number of prediction intervals. We have taken the case of $R = 0$ to compare our method to the best results of short-term prediction in number of deployments. However, note that in order to guarantee availability, it would be advisable to add a redundancy factor $R > 0$, therefore increasing the number of deployments.

Fig. 22 shows that the number of migrations required over time increases linearly, as expected, for the methods based on temporary deployments (with and without prediction). After one week, the short-term prediction-based method has performed nearly the same number of deployments than the bit vector-based with threshold 0.85. However, the latter does not perform any

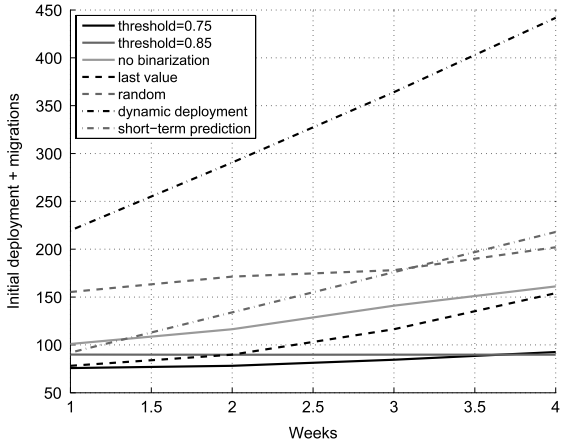


Fig. 22. Average number of migrations in different number of weeks after initial service deployment.

temporary deployments, so its value is constant over time. After 4 weeks, the low turn-over rate of the short-term prediction-based method ends up causing more deployments than the random policy.

In conclusion, we see that the use of long-term prediction generates fewer deployments, and therefore involves less communication cost, than short-term prediction when deploying long-lived services over volunteer resources.

8. Conclusion

Host availability prediction is critically important for increasing system availability and reliability as well as efficiency. This is especially true in systems based on non-dedicated resources, like volunteer computing, enterprise desktop grids and peer-to-peer systems. In such cases, communication costs associated with migration can be too high given the bandwidth of end hosts. Availability prediction can help reduce these costs.

We have presented a method for long-term availability prediction, and put it in practice in an availability-aware service deployment mechanism for volunteer computing systems. This mechanism has been inspired by the analysis of host availability in real system traces, and put to the test using these same real data.

Our simulations have shown that using the bit vectors to deploy a service by choosing a permanent set of machines to host it can provide high availability with relatively small redundancy. We have compared it to the following: (1) a similar method without the binarization step, (2) a method of choosing hosts available at the moment of deployment and (3) a method that simply chooses a random set of hosts for permanent deployment. The bit vector methods have performed better than all the rest.

Specifically, we have shown that, using 0.75 as a binarization threshold, the redundancy is lower and the availability higher than with the other approaches. Moreover, the required computations are simple and the mechanism only requires local information (i.e., the availability information of each host of a random subset of hosts), which makes it fitting for decentralized environments. It is also able to adapt to different types of hosts behavior, automatically selecting more nodes when their expected availability is low. Other mechanisms, like *last value* or *random*, require that the redundancy factor is manually adapted to the expected availability of hosts.

Using a threshold of 0.85 causes a higher redundancy, but gives extremely high availability guarantees. Therefore, in a real application the threshold and the redundancy factor used may be fine-tuned to whether increase the availability provided by the mechanisms or decrease the redundant resource usage.

Moreover, the long-term prediction-based techniques have also shown that they can obtain similar availability levels than those achieved using a short-term prediction technique [3], while requiring a lower number of deployments and migrations over time. This justifies the use of long-term availability prediction for deploying long-lived services and data, as opposed to short-term prediction which is more suited to task scheduling.

Work on long-term availability prediction can be further extended in some ways. For example, better mechanisms could be used to optimize the selection of hosts to have values as close to N as possible in every position of the sum vector. This could be done to minimize redundancy, and at the same time make good use of hosts with medium or low availability, contrary to the simple method presented which tends to favor hosts with high availability. Alternative ways to detect highly predictable hosts could also be used. However, these alternative mechanisms would probably require a larger amount of information, which is costly to obtain in a distributed system.

An unsolved problem is how to predict seasonal variations in host behavior. The presented mechanism could react to them by periodically re-evaluating deployments. A proactive solution, however, would require a lot of trace data, because seasonal effects use to appear over long periods of time (e.g. holidays). The main questions are how would a real system store and summarize this information, and how would this information be used, e.g. determining the initial deployment or rather detecting seasonal variations with short anticipation and then performing corrective actions.

Another related problem is prediction of permanent failures that can undermine durability of a deployment. This may be solved by simple methods like just removing a host from the pool whenever it has been disconnected for a long time. However, trace analysis could be used to validate this assumption or to propose other methods to guarantee durability.

Finally, we would like to test our prediction and deployment methodology using trace data of different systems. This is needed to see if our findings can be applied to other environments, or they are only valid with volunteer computing communities like SETI@home or systems with similar characteristics. This could be complemented with a real implementation and deployment over volunteer hosts.

Acknowledgments

We thank Lucas Schnorr for his invaluable aid in trace visualization. We thank David P. Anderson for making the SETI@home traces available.

This work has been partially supported by the "Comissionat per a Universitats i Recerca del Departament d'Innovació, Universitats i Empresa de la Generalitat de Catalunya" and by the "Fons Social Europeu" under grants FI and BE, and also by the HAROSA Knowledge Community of the Internet Interdisciplinary Institute (IN32009-AKC91). This work has also been carried out in part under the ANR project Clouds@home (ANR-09-JCJC-0056-01).

References

[1] 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.
[2] D. Anderson, 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.
[3] A. Andrzejak, D. Kondo, D.P. Anderson, Ensuring collective availability in volatile resource pools via forecasting, in: DSOM, 2008, pp. 149–161.
[4] R. Arpaci, A. Dusseau, A. Vahdat, L. Liu, T. Anderson, D. Patterson, The interaction of parallel and sequential workloads on a network of workstations, in: Proceedings of SIGMETRICS'95, 1995, pp. 267–278.

[5] M. Bakkaloglu, J.J. Wylie, C. Wang, G.R. Ganger, On correlated failures in survivable storage systems, Technical Report MU-CS-02-129, Carnegie Mellon University, May 2002.
[6] R. Bhagwan, S. Savage, G. Voelker, Understanding availability, in: Proceedings of IPTPS'03, 2003.
[7] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, G.M. Voelker, Total recall: system support for automated availability management, in: NSDI, 2004, pp. 337–350.
[8] W. Bolosky, J. Douceur, D. Ely, M. Theimer, Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs, in: Proceedings of SIGMETRICS, 2000.
[9] J. Chu, K. Labonte, B. Levine, Availability and locality measurements of peer-to-peer file systems, in: Proceedings of ITCom: Scalability and Traffic Control in IP Networks, 2003.
[10] F.L. Daniel Ford, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, S. Quinlan, Availability in globally distributed storage systems, in: Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, 2010.
[11] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, K. Ramchandran, Network coding for distributed storage systems, IEEE Transactions on Information Theory 56 (9) (2010) 4539–4551. doi:10.1109/TIT.2010.2054295.
[12] P. Dinda, Online prediction of the running time of tasks, Cluster Computing 5 (3) (2002) 225–236.
[13] P. Domingues, P. Marques, L. Silva, Resource usage of windows computer laboratories, in: Parallel Processing, 2005, ICPP 2005 Workshops, International Conference Workshops on, 2005, pp. 469–476. doi:10.1109/ICPPW.2005.77.
[14] J.R. Douceur, Is remote host availability governed by a universal law? SIGMETRICS Performance Evaluation Review 31 (3) (2003) 25–29.
[15] C. Elkan, Using the triangle inequality to accelerate *k*-means, in: ICML, 2003, pp. 147–153.
[16] T. Estrada, M. Taufer, K. Reed, Modeling job lifespan delays in volunteer computing projects, in: Cluster Computing and the Grid, 2009, CCGRID'09, 9th IEEE/ACM International Symposium on, 2009, pp. 331–338. doi:10.1109/CCGRID.2009.69.
[17] B. Javadi, D. Kondo, J. Vincent, D. Anderson, Mining for statistical availability models in large-scale distributed systems: an empirical study of seti@home, in: 17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS, 2009. URL: http://mescal.imag.fr/membres/derrick.kondo/pubs/javadi_mascots09.pdf.
[18] B. Javadi, D. Kondo, J. Vincent, D. Anderson, Discovering statistical models of availability in large distributed systems: An empirical study of SETI@home, IEEE Transactions on Parallel and Distributed Systems 22 (11) (2011) 1896–1903. doi:10.1109/TPDS.2011.50.
[19] P. Knežević, A. Wombacher, T. Risse, DHT-based self-adapting replication protocol for achieving high data availability, in: Advanced Internet Based Systems and Applications, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 201–210. doi:http://dx.doi.org/10.1007/978-3-642-01350-8_19 (Chapter). URL: http://dx.doi.org/10.1007/978-3-642-01350-8_19.
[20] D. Kondo, A. Andrzejak, D.P. Anderson, On correlated availability in Internet distributed systems, in: IEEE/ACM International Conference on Grid Computing, Grid, Tsukuba, Japan, 2008.
[21] D. Kondo, G. Fedak, F. Cappello, A.A. Chien, H. Casanova, Characterizing resource availability in enterprise desktop grids, Journal of Future Generation Computer Systems 23 (7) (2007) 888–903.
[22] D. Kondo, B. Javadi, A. Iosup, D. Epema, The failure trace archive: enabling comparative analysis of failures in diverse distributed systems, in: Cluster, Cloud and Grid Computing, CCGrid, 2010 10th IEEE/ACM International Conference on, 2010, pp. 398–407. doi:10.1109/CCGRID.2010.71.
[23] D. Kondo, M. Taufer, C. Brooks, H. Casanova, A. Chien, Characterizing and evaluating desktop grids: an empirical study, in: Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS'04, 2004.
[24] D. Long, A. Muir, R. Golding, A longitudinal survey of Internet host reliability, in: 14th Symposium on Reliable Distributed Systems, 1995, pp. 2–9.
[25] J.W. Mickens, B.D. Noble, Exploiting availability prediction in distributed systems, in: Proceedings of the 3rd Conference on Networked Systems Design & Implementation, 2006.
URL: http://www.eecs.umich.edu/~jmickens/predictors.pdf.
[26] Publications by BOINC projects. http://boinc.berkeley.edu/wiki/Publications_by_BOINC_projects.
[27] K. Ramachandran, H. Lutfiyya, M. Perry, Decentralized resource availability prediction for a desktop grid, in: Cluster, Cloud and Grid Computing, CCGrid, 2010 10th IEEE/ACM International Conference on, 2010, pp. 643–648. doi:10.1109/CCGRID.2010.54.
[28] F. Salfner, M. Lenk, M. Malek, A survey of online failure prediction methods, ACM Computing Surveys 42 (2010) 10:1–10:42. doi:http://doi.acm.org/10.1145/1670679.1670680 URL: http://doi.acm.org/10.1145/1670679.1670680.
[29] S. Saroiu, P. Gummadi, S. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of MMCN, 2002.
[30] L.M. Schnorr, A. Legrand, J.-M. Vincent, Visualization and detection of resource usage anomalies in large scale distributed systems, Tech. Rep. INRIA-00529569, INRIA, October 2010.
[31] W.T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, G. Gedye, D. Anderson, 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.
[32] R. Vilalta, C.V. Apte, J.L. Hellerstein, S. Ma, S.M. Weiss, Predictive algorithms in the management of computer systems, IBM Systems Journal 41 (2002) 461–474. doi:http://dx.doi.org/10.1147/sj.413.0461 URL: http://dx.doi.org/10.1147/sj.413.0461.
[33] D. Vyas, J. Subhlok, Volunteer computing on clusters, in: Proceedings of the 12th International Conference on Job Scheduling Strategies for Parallel Processing, JSSPP'06, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 161–175. URL: http://dl.acm.org/citation.cfm?id=1757044.1757052.
[34] R. Wolski, N. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, Journal of Future Generation Computer Systems 15 (5–6) (1999) 757–768.

Daniel Lázaro is a researcher at the Universitat Oberta de Catalunya. He holds a M.S. in Computer Science from the Universitat Politècnica de Catalunya and a Ph.D. by the Universitat Oberta de Catalunya. His research interests include scalable distributed algorithms and applications, service and task deployment in decentralized and cooperative environments and peer-to-peer and volunteer systems.

Derrick Kondo is a tenured research scientist at INRIA, France. He received his Bachelor's at Stanford University in 1999, and his Master's and Ph.D. at the University of California at San Diego in 2005, all in computer science.

His general research interests are in the areas of reliability, fault-tolerance, statistical analysis, scheduling and resource management. His current research interests are in the following areas: (1) failure and availability modeling of 100,000 + node systems (2) dynamic and cost-aware fault tolerance in Clouds (3) scheduling and resource management on unreliable and shared resources.

His research projects are supported by national, European, and industrial grants. He is co-founder of the Failure Trace Archive, which serves as a public repository of failure traces and algorithms for distributed systems.

Joan Manuel Marquès is an Associate Professor of Distributed Systems in the Computer Science Department at the Universitat Oberta de Catalunya as well as Researcher at the Internet Interdisciplinary Institute. His research interests include design of scalable and cooperative Internet services and applications, distributed computing and collaborative learning. Marquès has a Ph.D. in computer science from the Universitat Politècnica de Catalunya.