HAL
open science

Ensuring Collective Availability in Volatile Resource Pools Via Forecasting

Artur Andrzejak, Derrick Kondo, David P. Anderson

To cite this version:

Artur Andrzejak, Derrick Kondo, David P. Anderson. Ensuring Collective Availability in Volatile Resource Pools Via Forecasting. DSOM, 2008, Unknown, pp.149-161. hal-00953615

HAL Id: hal-00953615

https://inria.hal.science/hal-00953615v1

Submitted on 25 Mar 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Ensuring Collective Availability in Volatile Resource Pools Via Forecasting*

Artur Andrzejak[1], Derrick Kondo[2], and David P. Anderson[3]

[1] ZIB, Germany
andrzejak@zib.de
[2] INRIA, France
dkondo@imag.fr
[3] UC Berkeley, USA
davea@ssl.berkeley.edu

Abstract. Increasingly services are being deployed over large-scale computational and storage infrastructures. To meet ever-increasing computational demands and to reduce both hardware and system administration costs, these infrastructures have begun to include Internet resources distributed over enterprise and residential broadband networks. As these infrastructures increase in scale to hundreds of thousands to millions of resources, issues of resource availability and service reliability inevitably emerge. Our goal in this study is to determine and evaluate predictive methods that ensure the availability of a *collection* of resources. We gather real-world availability data from over 48,000 Internet hosts participating in the SETI@home project. With this trace data, we show how to reliably and efficiently predict that a collection of N hosts will be available for T time. The results indicate that by using replication it is feasible to deploy enterprise services or applications even on such volatile resource pools.

1 Introduction

Services are being deployed increasingly over large-scale computational and storage architectures. Internet services execute over enormous data warehouses (such as Google) and cloud computing systems (such as Amazon's EC2, S3). Scientific services are deployed over large-scale computing infrastructures (such as TeraGrid, EGEE).

To meet ever-increasing computational demands and to reduce both hardware and system administration costs, these infrastructures have begun to include Internet resources distributed over enterprise and residential broadband networks. Enterprises, such as France Telecom [1], are currently deploying a video-encoding service where the computational load is distributed among the peers in a residential broadband network. Universities, through academic projects such as

* This research work is carried out in part under the projects CoreGRID (Contract IST-2002-004265) and SELFMAN (contract 34084), both funded by the EC.

F. De Turck, W. Kellerer, and G. Kormentzas (Eds.): DSOM 2008, LNCS 5273, pp. 149–161, 2008.
© IFIP International Federation for Information Processing 2008

Folding@home [2], deploy scientific applications that use PetaFLOPS of computing power from Internet resources.

Infrastructures of this scale are exposed to issues of availability and reliability. Low mean-time-to-failures (MTTF) and entire system crashes have plagued both service providers and customers. For example, when Amazon's S3 storage serviced crashed [3], numerous companies that depended on Amazon's services were stranded. In Internet environments, such as BOINC, resource unavailability can be as high as 50% [4].

At the same time, resource availability is critical for the reliability and responsiveness (low response latency) of services. Groups of available resources are often required to execute tightly-coupled distributed and parallel algorithms of services. Moreover, load spikes observed with Internet services (such as the commonly observed slashdot effect [5]) require guarantees that a collection of resources is available.

Given this need, our goal in this study is to determine and evaluate predictive methods that ensure the availability of a *collection* of resources. We strive to achieve collective availability from Internet distributed resources, arguably the most unreliable type of resource world-wide. Our predictive methods could work in coordination with a virtualization layer for masking resource unavailability.

Specifically, the contributions of this study are the following:

- We determine accurate methods and parameters for predicting resource availability. In particular, we investigate the factors that influence prediction error and determine indicators of resource predictability.
- We show how these methods and indicators can be used for predicting the availability of groups of resources and the associated costs. In particular, via trace-driven simulation, we evaluate our prediction method for collections of resources with different predictability levels. Our performance metrics include the success of predictions, and the cost in terms of redundancy and migration overhead for fault-tolerance.

Paper structure. In Section 2 we describe the data used in our study. Section 3 is devoted to the concept of predictability estimation, while Section 4 describes and evaluates the simulation approach used to ensure collective availability. Section 5 discusses related work. We conclude with Section 6.

2 Measurement Method

Our approach for gathering measurement data at a large-scale was to use the Berkeley Open Infrastructure for Network Computing (BOINC) [6]. BOINC serves as the underlying software infrastructure for projects such as SETI@home [7] and is deployed currently 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). The factors that can cause CPU unavailability include machines failures, power cycling, and user load. 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. The meaning 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 [4], and experience in [8] have shown this to be a good approximation of availability on real platforms.

This modified BOINC client was then made available for download starting on April 1, 2007. After a trace collection period of about seven months, the log files of these hosts were collected at the BOINC server for SETI@home on February 12, 2008. By December 1, 2007 more than 48,000 hosts had downloaded and were running the modified client. (By the end date, we had collected data from among 112,268 hosts, and the logs traced 16,293 years of CPU availability.) We use the subset of hosts (>48,000) that were actively running the client on December 1, 2007, and use the trace data for these hosts up to the end date of February 12, 2008.

Our trace data also includes the demographics of hosts, when specified by the BOINC user. About 32,000 hosts had specified host types. Of these hosts, about 81% are at home, 17% are at work, and 2% are at school.

3 Estimating Predictability and Forecasting Availability

This section focuses on forecasting the availability of individual hosts along with the estimation of the forecast accuracy. Our prediction methods are measurement-based, that is, given a set of availability traces called *training data* we create a predictive model of availability that is tested for accuracy with the subsequent (i.e. more recent) *test data*. For the sake of simplicity we refrain from periodic model updates.

An essential parameter for successful host selection is the expected accuracy of prediction (w.r.t. the test data). We designate it as *(host) predictability*. An essential ingredient of our approach is that we compute estimators of predictability from the training data *alone*. As we will show in Section 4, using this metric for host selection ensures higher availability ratios. Together with average availability, this metric allows also for fast elimination of hosts for which the predictions are less accurate and thus less useful.

3.1 Prediction Methodology and Setup

We compute for each host a predictive model implemented as a Naïve Bayes classifier [9]. A classification algorithm is usually the most suitable model type if inputs and outputs are discrete [10] and allows the incorporation of multiple inputs and arbitrary *features*, i.e., functions of data which expose better its information content. We have also tried other algorithms such as decision trees with comparable accuracy results. Since it is known that (given the same inputs and prior knowledge) no predictive method performs significantly better than others [11,12], we stick to this computationally efficient classifier.

Each sample in the training and test data corresponds to one hour and so it can be represented as a binary (01) string. Assuming that a prediction is computed at time T (i.e. it uses any data up to time T but not beyond it), we attempt to predict the complete availability versus (complete or partial) non-availability for the whole *prediction interval* $[T, T+p]$. The value of p is designated as the *prediction interval length* (*pil*) and takes values in whole hours (i.e. $1, 2, \ldots$). To quantify the prediction accuracy evaluated on a set S of such prediction intervals we use the ratio (called *prediction error*) of mispredicted intervals in S to $|S|$. The value of the parameter *pil* influences strongly the prediction accuracy. As other factors likely affect accuracy, we have also studied the length of the training data interval and the host type (i.e. deployed at home, work, or school).

To help a classifier, we enrich the original 01 data with features from the following groups. The *time* features include for each sample calendar information such as hour in day, hour in week, day in week, day in month etc. The *hist* features are (for each sample) the sums of the recent k "history bits" for $k = 2, 5, 10, 20, 50$ and 100. They express information about the length of the current availability status and the availability average over different periods.

3.2 Types of Predictable Patterns

There are several types of availability patterns that make a host predictable [13,14]. As a *type A* we understand behavior with long (as compared to *pil*) consecutive samples (stretches) of either availability or non-availability. Patterns of *type B* feature periodic or calendar effects, e.g. diurnal availability. For example, host availability over weekends or nights would create patterns of the later kind. Finally, availability of a host might occur after specific availability patterns (like alternating on/off status etc.). We disregard the last type as it is unlikely to occur in our scenario.

Knowing the predominant pattern type is helpful in designing the predictability indicators. For type A, simple metrics like the average length of an availability run might suffice, while type B requires more sophisticated methods like Fast Fourier Transformation (FFT). We identify the predominant patterns type by using different groups of features (from *hist* and *time*) in predictions and comparing the prediction accuracy. Obviously the *hist* features would lead to higher accuracy for type A patterns, while the *time* features are likely to be useful in presence of type B patterns.

3.3 Predictability Indicators

We have implemented as the predictability indicators a variety of metrics which are likely to indicate patterns of type A, B and possibly others. All indicators are computed over the training data only. The *aveAva* is the average host availability in the training data. The *aveAvaRun* (*aveNavaRun*) is the average length of a consecutive availability (non-availability) run. The *aveSwitches* indicator is the average number of changes of the availability status per week.

The last two indicators *zipPred* and *modelPred* are more involved and computationally costly. The former is inspired by [11] and is essentially the reciprocal value of the length of a file with the training data compressed by the Lempel-Ziv-Welch algorithm (raw, without the *time* and *hist* features). The rationale is that a random bit string is hardly compressible while a bit string with a lot of regularities is. To compute the last indicator, we train the classifier on half of the training data and compute the classification error (as above) on the other half. The *modelPred* value is then the reciprocal value of this error.

To compare indicators in their power to estimate the prediction error we compute the Spearman's rank correlation coefficient over all hosts for different *pil* values. We also verify the correlations by visual inspection of scatter plots (Figure 3).

3.4 Implementation and Running Time

The prediction and simulation experiments have been implemented and conducted using the Matlab 2007b environment. This framework was complemented by the PRTools4, a Matlab toolbox for pattern recognition [15]. As the Naïve Bayes classifier for predictions we used the `naivebc` method of PRTools4 with the default parameters.

Since Matlab is a partially interpreted language the running times of the algorithms are only upper bounds of tailored implementations. However, a complete experimental evaluation of a single host (including file operations, computation of *all* predictability indicators, classifier training and approximately 330 predictions in the test interval) required on average 0.25 seconds on a 2.0 GHz Xeon machine under Linux.

Even if this time is negligible compared to a typical *pil* value, an efficient, tailored implementation is likely to be much faster. Assuming that the length of the training interval in hours (i.e. the length of the 01 training string) is n, the upper bounds on the number of operations (per host) are as follows. The computation of *aveSwitches* is $O(n)$ with constant of 1 (other predictability indicators are not used in a production scenario). An efficient computation of the features *hist* and *time* requires time $O(n)$ with a constant below 10 in each case. An implementation of the Naïve Bayes classifier which precomputes all conditional probabilities during training requires time $O(nd)$ for training and $O(2d)$ for a prediction, where d is the number of features (below 20 in our case). As an example, assuming a training interval of 30 days ($n = 720$) and $d = 20$, a one-time feature computationa and training would require below $3 \cdot 10^4$ operations, and a single prediction about 40 operations. Note such a training is performed only if a host has passed the *aveSwitches* test which itself costs merely about $n = 720$ operations.

3.5 Experimental Evaluation

If not otherwise stated, the experiments used all the data described in Section 2 and the size of the training data was 30 days.

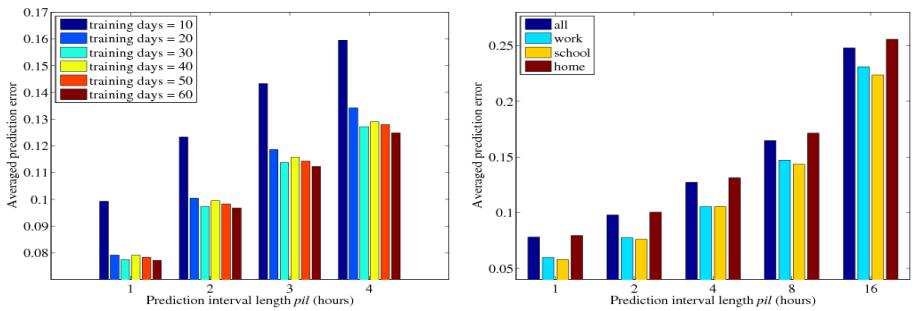


Fig. 1. Prediction error depending on the training data length and *pil* (left); Prediction error depending on the host type and *pil* (right)

Factors influencing the prediction error. Figure 1 (left) shows the dependence of the prediction error on the length of the training data and the *pil* value for a subset of 10,000 randomly selected hosts. While the error decreases significantly if the amount of training data increases from 10 to 20 days, further improvements are marginal. We have therefore used 30 days as the amount of training data for the remainder of this paper. Figure 1 (right) illustrates that the host type influences consistently the prediction error, with work and school hosts being more predictable. Despite of this effect, the simulation results did not show any significant differences between host types which can be attributed to low magnitude of differences.

Both figures show a strong influence of the prediction interval length, *pil*, on the prediction error. This is a consequence of increased uncertainty over longer prediction periods and the "asymmetric" definition of availability in the prediction interval (a short and likely random intermittent unavailability makes the whole interval unavailable).

Types of availability patterns. As described in Section 3.2 we measured the prediction error depending on groups of used data features for a group of

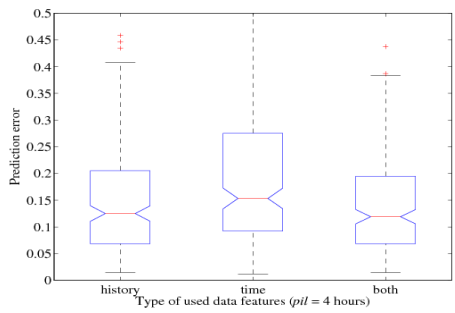


Fig. 2. Dependency of the prediction error on the data features

3000 randomly selected hosts. Figure 2 shows the lower quartile, median, and upper quartile values in the box while whiskers extend 1.5 times the interquartile range from the ends of the box. The median error is largest when using only *time* features and smallest when using both feature groups. While the magnitude of the difference is low, the relative order of cases was consistent across all *pil* values and host types. We conclude that most predictable hosts have availability patterns of type A (long availability state stretches) with few hosts exhibiting periodic or calendar predictability. Despite of this finding, we have included both the *time* and *hist* features in the classifier training as the computation of the them requires only linear time in the length of the training interval.

Evaluating predictability indicators. Table 1 shows correlations between prediction error and various predictability indicators defined in Section 3.3 (rows correspond to different prediction interval lengths). The strongest and most consistent correlation values has the *modelPred* indicator. However, as it is computationally most expensive, we identified the *aveSwitches* indicator as a viable substitute. For $pil = 1, 2, 4$ its correlation is comparable to *modelPred,* however it becomes much weaker for $pil = 8$ and 16. We could not fully explain this phenomenon, especially since the simulation results confirm its good quality even for these high *pil* values. However, a visual inspection of scatter plots for $pil = 2$ and $pil = 16$ (Figure 3) reveals that while for $pil = 2$ the correlation is obvious, for the higher *pil* value a relationship between the indicator and prediction error still exists but it is blurred by many "outliers". Finally, Table 1 confirms the finding that there is no clear relationship between average availability *aveAva* and the prediction error.

4 Evaluation of Group Availability Prediction Via Simulation

4.1 Method

We conducted trace-driven simulation applying the predictor determined in the previous sections. The predictor uses a training length of 30 days, which was shown to minimize prediction error according to Figure 1. This predictor is used to determine groups of available hosts, and the quality of the prediction is evaluated in simulation.

Table 1. Spearman's rank correlation coefficient between prediction error and various predictability indicators (rows correspond to different *pil* values)

pil	***aveAva***	***aveAvaRun***	***aveNavaRun***	***aveSwitches***	***zipPred***	***modelPred***
1	-0.370	-0.594	0.085	0.707	-0.654	-0.724
2	-0.275	-0.486	-0.011	0.678	-0.632	-0.690
4	-0.119	-0.303	-0.119	0.548	-0.502	-0.640
8	0.194	0.056	-0.245	0.195	-0.127	-0.642
16	0.211	0.091	-0.185	0.057	0.062	-0.568

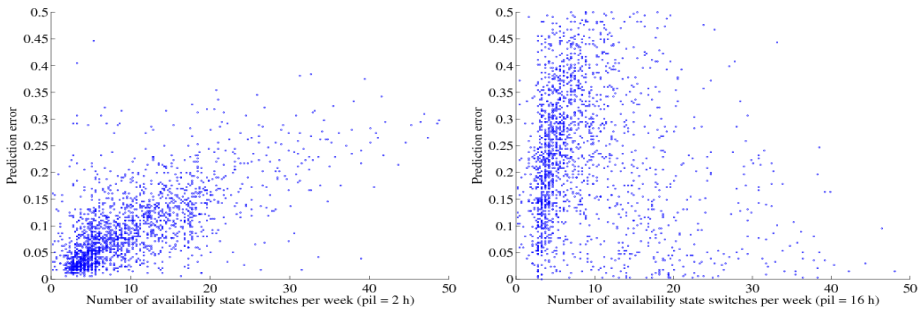


Fig. 3. Scatter plots of number of availability state changes per week (*aveSwitches*) and the prediction error (random subset of 2000 hosts)

We divide the hosts into groups based on the best predictability indicator described in Section 1, namely *aveSwitches*. To determine how to divide the hosts, we plotted the distribution of the *aveSwitches* values among the hosts (see Figure 4). The point (x, y) in Figure 4 means that y fraction of the hosts have an *aveSwitches* value of x or greater. Given the skew of this curve, we choose the median value of 7.47 near the "knee" of the curve to divide the hosts into two groups with high and low predictability respectively.

The parameters for simulation include the number of hosts desired to be available for some time period which is potentially longer than *pil*. In this case one could simply use the predictor at the end of each *pil* interval and repeat until the desired time period is reached. We refer to the total desired time period of availability as the *threshold* (which is some multiple of *pil*). For example, if the *pil* is 4 but the threshold is 12, we would rerun the predictor after time intervals of 4 and 8 elapse.

Another parameter is *redundancy*. We define redundancy to be $(R - N)/N$ where N is the number of hosts desired to be available, and R is the number of hosts actually used. The number of hosts we use in simulation are 1, 4, 16, 64, 256, 1024. The redundancy is in the range of [0, 0.50]. Due to space limitations, the simulation results we present below are for a *pil* of 4, though we also tried other *pil* values and observed similar trends.

The predictor is run using a test period of about two weeks which follow directly the training period. For each data point shown in the figures below, we ran about 30,000 simulations to ensure the statistical confidence of our results. In total, we executed more than 2 million simulations.

4.2 Performance Metrics

We measured the performance of the predictions in simulation by a *success rate* metric. In a simulation trial, we randomly choose R number of hosts from the pool predicted to be available for the entire threshold. We run trials in this way throughout the test period. We then count the number of trials where the number

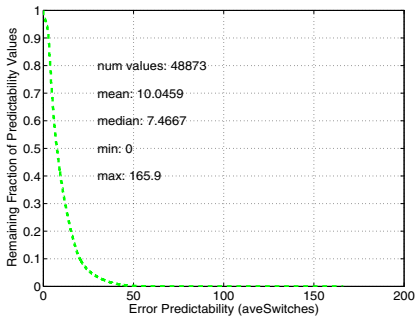


Fig. 4. Distribution of *aveSwitches*

of hosts actually available A (out of R) is greater than or equal to the desired number N. If A is greater than or equal to N for the entire threshold, then we consider the simulation trial a success. Otherwise, it is considered a failure. This fraction of the number of successes to total number of trials is defined as the success rate.

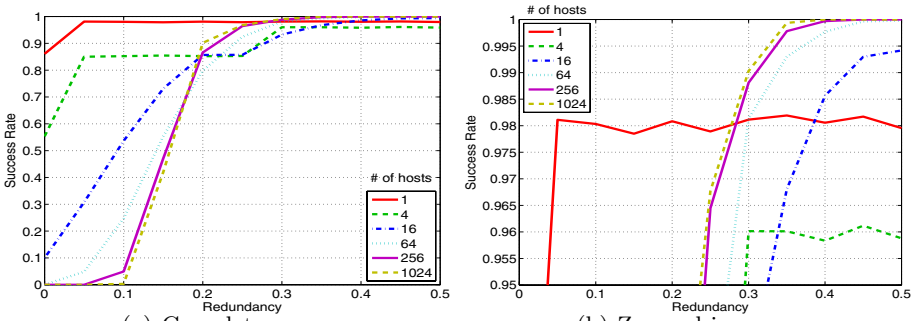
We also measure performance by the host *turnover rate* for thresholds greater in length than the *pil*. The turnover rate indicates the overheads (due to process migration, or service re-deployment, for example) due to changes in the predicted state of hosts from one sub-threshold to the next. We computed the turnover rate by first determining an active set of R hosts predicted to be available during some sub-threshold of length *pil*. In the following sub-threshold, we determine which hosts in the active set are predicted to be *unavailable*. The fraction of hosts in the active set that change from an available state to an unavailable state from one sub-threshold to the next is the turnover rate. Our method for computing the turnover rate gives an upper bound on fault-tolerance overheads.

We then compute the average turnover rate across all sub-thresholds of size *pil* throughout the entire test period. We conduct this process for active sets with different numbers of hosts and also different hosts randomly chosen from the pool of hosts predicted to be available.

4.3 Results

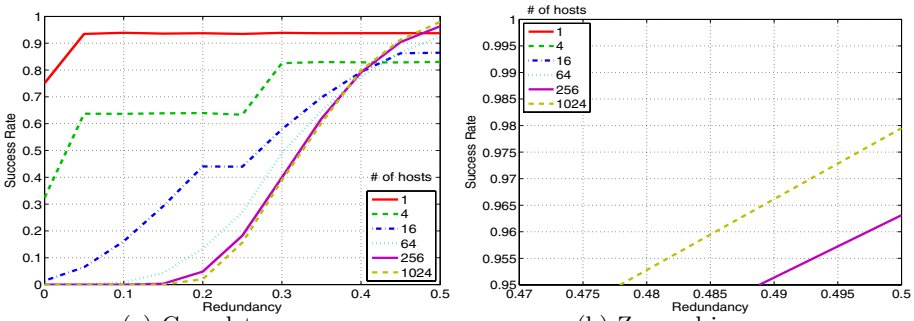
In Figures 5 and 6, we show the success rate achieved for various levels of redundancy and number of hosts desired. Figure 5 focuses on the high predictability group, and Figure 6 focuses on the low predictability group. In each of the subfigures, we show the results in the entire range (left), and also for the zoomed-in range (right) for success rates in [0.95,1.00] or smaller.

In Figure 5 (left), we observe that when redundancy is 0, the success rate decreases exponentially with the number of hosts desired. However, as redundancy increases, the success rate increases dramatically as well. We observe the redundancy necessary to achieve success rates of 0.95 or higher in Figure 5 (right). In particular, if we look at the redundancy where the success rate is 0.95, we find that redundancy of 0.35 can achieve 0.95 success rates for groups up to 1024 in size.



(a) Complete range (b) Zoomed-in range

Fig. 5. Success rates versus redundancy for high predictability group and *pil* 4

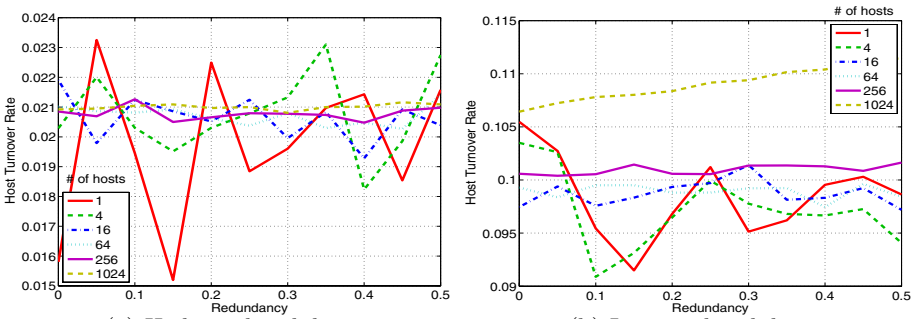


(a) Complete range (b) Zoomed-in range

Fig. 6. Success rates versus redundancy for low predictability group and *pil* 4

In Figure 6, we observe similar trends in terms of success rate versus redundancy. However, we observe clear differences between the high and low predictability groups in the amount of redundancy needed to achieve the same level of group availability. For example, with the high predictability group, a redundancy of 0.35 will achieve success rates of 0.95 or higher. With the low predictability group, only the groups with 256 and 1024 desired hosts can achieve the same level of success rates; at the same time, high redundancy greater than 0.45 is required. Thus grouping hosts by predictability levels using the *aveSwitches* indicator significantly improves the quality of group availability prediction, and consequently the efficiency of service deployment.

Services will sometimes need durations of availability longer then the *pil*. In these cases, one can just reevaluate the prediction at the beginning of a new prediction interval, i.e., sub-threshold. The question is what are the costs in terms of service re-deployment across sub-thresholds. In Figure 7, we observe turnover rates as a function of redundancy and the number of hosts. We observe that the turnover rates are relatively low. The turnover rate is less than 0.024 for the high predictability group, and about 0.115 for the low predictability group.



(a) High predictability (b) Low predictability

Fig. 7. Turnover rates versus redundancy for *pil* 4

This shows that the overheads (of process migration, or using a checkpoint server, for example) with thresholds larger than the *pil* are relatively low. Observe that the stated turnover rates are for two consecutive *pil* intervals. The *cumulative* turnover rate for the whole threshold is $r * n$ where n is the number of sub-thresholds and r is the turnover rate.

Another important aspect of Figure 7 is the relationship between the number of hosts and turnover rate. We observe that the host turnover rate does not increase dramatically with the desired number of hosts. Instead, it increases only by a few percent even when the number of hosts increases by a factor of 4. This indicates that turnover rate scales with an increase in number of hosts. We also computed the turnover rates with a *pil* of 2. We found similar trends, and and the turnover rates were either equal or an order of magnitude lower.

5 Related Work

Forecasting is an established technique in the arsenal of proactive management of individual computer systems, with applications ranging from capacity estimation to failure prediction [10,12]. This study differs from other prediction studies in three main respects, namely types of hosts considered (inclusion of home desktops versus only those in the enterprise), the type of measurements used for the evaluation of prediction (CPU availability versus host availability), and the prediction issues investigated (the accuracy of efficient and scalable prediction methods, indicators of predictability, and the use and overheads of prediction of group availability).

With respect to the types of measurements, our data set consists of traces of CPU availability, which is a stricter and more accurate representation of resource's true availability. Most (P2P) availability studies [16,17,14] and prediction studies [14] focus on host availability, i.e., whether the host is pingable or not, instead of CPU availability. However, resources can clearly have 100% host availability but 0% CPU availability. Paradoxically, in our more volatile

system, we find that simple prediction methods are suitable when applied with predictability indicators.

Moreover, with respect to the types of hosts characterized, the studies in [18,4] consider only resources in the enterprise (versus in home) environments. By contrast, the majority of resources in our study lie on residential broadband networks. Again, the time dynamics of CPU availability of home resources differ from those in enterprises. For example, the mean availability lengths found in this study are about 5.25 times greater than those in enterprise environments[4]. Also, the mean fraction of time that a host is available is about 1.3 times lower than that observed in enterprise desktop grids[4]. The studies in [19] and [20] focused solely on enterprise environments. For example, the Resource Prediction System (RPS) [19] is a toolkit for designing, building and evaluating forecast support in clusters. The Network Weather Service (NWS) is another well-known system for predicting availability in Grids (composed of multiple clusters).

With respect to prediction issues studied, we focus on novel prediction issues compared to previous works [14,19,20]. We focus on simple, scalable forecasting methods coupled with an efficient approach to filter out non-predictable hosts. Furthermore, we adjust our methods to types of predictable availability patterns for enabling group availability prediction.

6 Conclusions

In in this paper we attempted to show that a deployment of enterprise services in a pool of volatile resources is possible and incurs reasonable overheads. The specific technical contributions are this paper were as follows:

- We showed that the primary reason for the predictability of certain Internet hosts is the existence of long stretches of availability, and such patterns can be modeled efficiently with basic classification algorithms.
- We also demonstrated that simple and computationally cheap metrics are reliable indicators of predictability, and that resources could be divided into high and low predictability groups based on such indicators.
- For the high predictability group, via trace-driven simulation, we found that our prediction method can achieve 95% or greater success with collections of resources up to 1,024 in size using redundancy levels of 35% or less.
- For the high and low predictability groups, we found that the host turnover rates are less than 2.4% and 11.5% respectively. This indicates that prediction across long thresholds with low overheads is possible.

As a future work we plan to extend our experiments to other host pools, including PlanetLab. We also intend to study whether including additional features and inputs (such as CPU, memory or network utilization) can improve the prediction accuracy. Another research goal is to refine the predictability groups beyond low and high types.

References

1. Krishnaswamy, R.: Grid4all, http://www.grid4all.eu
2. Larson, S.M., Snow, C.D., Shirts, M., Pande, V.S.: Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. Computational Genomics (2003)
3. Carr, N.: Crash: Amazon's s3 utility goes down, http://www.roughtype.com
4. Kondo, D., et al.: Characterizing and Evaluating Desktop Grids: An Empirical Study. In: Proceedings of the IPDPS 2004 (April 2004)
5. Adler, S.: The slashdot effect: An analysis of three internet publications, http://ldp.dvo.ru/LDP/LG/issue38/adler1.html
6. Anderson, D.P.: BOINC: A system for public-resource computing and storage. In: Buyya, R. (ed.) Proceedings of 5th International Workshop on Grid Computing (GRID 2004), Pittsburgh, PA, USA, November 8, 2004, pp. 4–10. IEEE Computer Society, Los Alamitos (2004)
7. Sullivan, W.T., Werthimer, D., Bowyer, S., Cobb, J., Gedye, G., Anderson, D.: 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)
8. Malecot, P., Kondo, D., Fedak, G.: Xtremlab: A system for characterizing internet desktop grids (abstract). In: Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing (2006)
9. John, G.H., Langley, P.: Estimating continuous distributions in Bayesian classifiers. In: Proc. 11th Conference on Uncertainty in Artificial Intelligence, pp. 338–345. Morgan Kaufmann, San Francisco (1995)
10. Vilalta, R., Apte, C.V., Hellerstein, J.L., Ma, S., Weiss, S.M.: Predictive algorithms in the management of computer systems. IBM Systems Journal 41(3), 461–474 (2002)
11. Keogh, E.J., Lonardi, S., Ratanamahatana, C.A.: Towards parameter-free data mining. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 206–215 (August 2004)
12. Andrzejak, A., Silva, L.: Using machine learning for non-intrusive modeling and prediction of software aging. In: IEEE/IFIP Network Operations & Management Symposium (NOMS 2008), April 7-11 (2008)
13. Douceur, J.R.: Is remote host availability governed by a universal law? SIGMETRICS Performance Evaluation Review 31(3), 25–29 (2003)
14. Mickens, J.W., Noble, B.D.: Exploiting availability prediction in distributed systems. In: NSDI, USENIX (2006)
15. van der Heijden, F., Duin, R.P.W., de Ridder, D., Tax, D.M.J.: Classification, Parameter Estimation and State Estimation. John Wiley & Sons, Chichester (2004)
16. Bhagwan, R., Savage, S., Voelker, G.: Understanding Availability. In: Proceedings of IPTPS 2003 (2003)
17. Saroiu, S., Gummadi, P., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proceedinsg of MMCN (January 2002)
18. Bolosky, W., Douceur, J., Ely, D., Theimer, M.: Feasibility of a Serverless Distributed file System Deployed on an Existing Set of Desktop PCs. In: Proceedings of SIGMETRICS (2000)
19. Dinda, P.: A prediction-based real-time scheduling advisor. In: 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), vol. 10 (April 2002)
20. Wolski, R., Spring, N., Hayes, J.: Predicting the CPU Availability of Time-shared Unix Systems. In: Proceedings of 8th IEEE High Performance Distributed Computing Conference (HPDC8) (August 1999)