Chapter 1

Modeling and Optimizing Availability of Non-Dedicated Resources

Artur Andrzejak

Heidelberg University

Derrick Kondo

INRIA

1.1 Introduction

An inherent drawback of non-dedicated computing resources is low and uncontrollable availability of individual hosts. This phenomenon limits severely the range of scenarios in which such resources can be used and adds a significant deployment overhead. Approaches to compensate for these difficulties use redundancy-based fault-tolerance techniques supported by modeling and prediction of availability. In the first part of this chapter we discuss a variety of modeling techniques ranging from probability distributions to machine learning-based prediction techniques. Subsequently we focus on methods to provide resource-efficient and cost-minimizing fault-tolerance. Here redundancy is mandatory to mask the outages of individual machines, yet on the other hand it might increase overhead and resource cost. We describe how availability models help here to obtain statistical guarantees of (collective)

		Number of hosts n								
		2	**3**	**4**	**5**	**6**	**8**	**10**	**16**	**32**
$p_i \geq$	**0.1**	0.19	0.27	0.34	0.41	0.47	0.57	0.65	0.81	0.97
	0.2	0.36	0.49	0.59	0.67	0.74	0.83	0.89	0.97	1.00
	0.4	0.64	0.78	0.87	0.92	0.95	0.98	0.99	1.00	1.00

Table 1.1: Lower bounds on probability of a failure of *at least one* host in a group of n hosts (with individual failure probabilities of $p_1, \ldots, p_n$)

availability and how total costs of the resources can be balanced against reliability properties. We also consider the issue of adjusting application architectures in order to tolerate partial resource failures. This promises to broaden the type of applications deployed on voluntarily computing resources from embarrassingly parallel jobs to Map-Reduce-type applications or even web services.

1.1.1 Problem of individual failures

Outages of individual resources occur in all types of distributed systems. In dedicated environments, such an outage is usually caused by a hardware or software *failure*. To simplify our considerations, we will use the term failure when referring to outage due to such a malfunction or because the resource owner has interrupted a "non-dedicated" execution (e.g. by shutting down his computer). The essential difference between dedicated and non-dedicated scenarios is that in the latter case the probability of a failure (within a time unit) is typically much higher than of dedicated hosts.

Consider a collection of n hosts where each can fail (independently of others) with a probability p_i within a time interval T. Then the probability that *at least one* host fails within T is

$$1 - \prod_{i=1}^{n}(1 - p_i). \tag{1.1}$$

Assuming $p_i \geq 0.2$ for T of e.g. two hours (a realistic assumption e.g. in case of SETI@home hosts [1]) Table 1.1 shows that already for 4 hosts a chance that at least one host fails within T surpasses 50% (it is at least 0.59). This high failure probability has severe consequences on the type of applications which can be executed on non-dedicated hosts. Most importantly, it shows that without additional measures (such as redundancy and checkpointing) it is virtually impossible in such environments to execute any parallel applications, i.e. applications which assume *synchronized availability* of several (distributed) resources.

Consequently, current applications running on non-dedicated resources are predominantly embarrassingly-parallel computations where each host processes a chunk of data (or a parameter configuration) independently and does

not communicate with other workers. But even in the latter scenario a failure might have several negative effects, including:

- lost of data - this is a major problem in systems such as Wuala which use non-dedicated (along with dedicated) resources for data storage
- lost computation - in case of computational jobs this includes lost of work since last checkpoint; also there is the case where all work since last upload of the results is lost (if hosts drops out permanently)
- delayed job completion - as the work performed by a host in a time unit is a product of its computing capacity and its average availability, each failure implies less work in a time unit and so a possible delay of the completion
- degradation of overall service availability - if hosts are used in a "service" scenario (as in the case of Wuala), individual outages can lead to an overall failure of a (replicated) service
- need for replacement / migration actions - temporarily or permanently failed hosts trigger need for data migration and replication actions, as well as afresh scheduling actions.

Obviously, even non-parallel computations require a high degree of redundancy, checkpointing and other mechanisms if hosted on non-dedicated resources. This significantly increases the overhead and reduces by a considerable factor the overall deployment efficiency along with the computational capacity of such infrastructures.

As mentioned above, cases where unexpected failures of individual resources are common and need to be treated are not limited to non-dedicated resources. An example are low-cost computational nodes deployed in large-scale data centers as in case of Google infrastructures. In Equation (1.1) large n implies that the probability of at least one failure becomes high even if probability of a single node failure is low. Another example are recently introduced Spot Instances in the Amazon Elastic Compute Cloud (EC2) [2]. These instances (typically spare capacity of EC2) can be revoked *abruptly* due to price and demand fluctuations. While offering lower resource costs their availability behavior introduces the same problems as in case of non-dedicated hosts.

We see that guaranteeing stable and efficient operation of a system composed of either unreliable or many resources is a problem transcending the domain of voluntarily computing. The concepts and methods presented in this chapter attempt to be sufficiently generic to cover this extended set of scenarios.

1.1.2 Chapter contents

The focus of this chapter is modeling of availability of individual resources and their collections. For availability optimization we turn our attention solely to *collections* of resources. There are several reasons for this:

- While it is possible and beneficial to model availability of individual resources (see Section 1.2.2) we have little power (except for host filtering) to enforce the availability of *specific* hosts.
- Tightly-coupled distributed, parallel algorithms and many services can be only provisioned by resource collections.
- Moreover, even in case of embarrassingly parallel computations we need to provide redundancy and understand its amount for reasons of efficiency. Here again it is necessary to consider availability optimization in collections of resources.

In Section 1.2 we present methods for modeling availability of non-dedicated resources on short-term and long-term time scales. Section 1.3 considers approaches for enhancing availability of resource collections. Several aspects are considered: trade-offs between redundancy and availability guarantees and monetary costs of such guarantees if non-dedicated resources are "enriched" by dedicated (cloud computing-type) resources. Section 1.4 illustrates these methods on empirical studies performed on a large number of SETI@home hosts. The final Section 1.5 contains the conclusions.

1.2 Modeling Availability of Non-Dedicated Resources

Modeling of host availability behavior serves two goals:

- *Analyzing past availability behavior* of individual hosts and their groups. The results can be applied in a multitude of ways, including: estimation of cumulative computational capacity; understanding availability and failure rate distributions (or, equivalently knowing share of hosts with average availability / failure rate in a given interval); discovering periodic temporal patterns of availability and long-term trends; filtering of hosts by specific criteria like "stable behavior"; detection of large-scale anomalies or failures such as partial network outages.
- *Predicting short-term availability* of individual hosts or their groups, where short-term means one to several hours. This can be directly applied for availability enhancement discussed in Section 1.3. As detailed below, predictive methods frequently use models of *past* behavior. Also

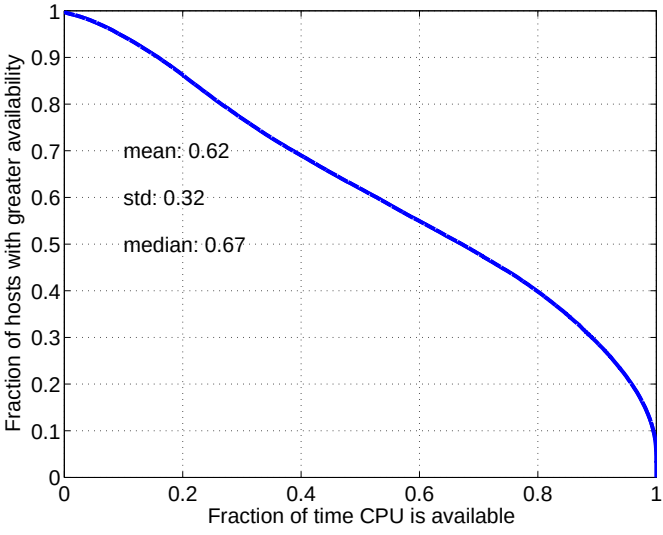

Figure 1.1: CPU availability of SETI@home hosts

note that long-term availability prediction (beyond one day) is usually not feasible except for a small class of hosts with very "regular" behavior.

Obviously considering more input data and using more sophisticated availability models is likely to yield more precise results (under the assumption that correct, i.e. non-overfitting [3] approaches are used). In general, we consider the following classes of availability models with increasing level of sophistication:

- cumulative models such as distributions of average availability and time between failures (Section 1.2.1)
- individual models which capture availability behavior of a single host (Section 1.2.2)
- models which consider individual behavior as well as their dependencies (e.g. short-term correlations between host behavior).

The last case still awaits an empirical study and is a challenge in terms of computational complexity for the amount of hosts considered in Section 1.4.

1.2.1 Availability distributions

One of the fundamental forms of availability modeling is to consider the fraction of time when each host is available ($aveAva$) and to compute the (cumulative) distribution function of this metric. The distribution is essentially

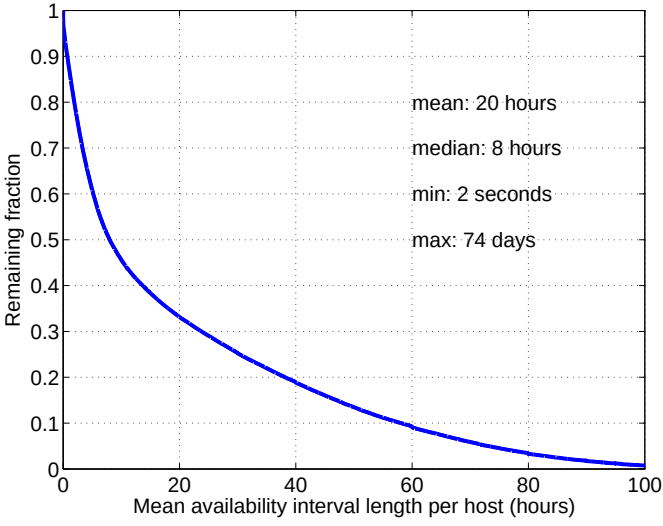

Figure 1.2: Average availability run per host in hours

computed by sorting the hosts according to this metric. Figure 1.1 (taken from [4]) shows a plot of such a distribution for a subset of 112,268 hosts participating in the SETI@home project between April 1, 2007 and February 12, 2008 (see [4]); here CPU availability is understood as host availability. Obviously 40% of hosts are available at least 80% of the time while almost 70% of hosts is available at least 40% of time.

Even if each host is characterized by only a single metric, their distribution allows us to answer several important questions: What is the cumulative processing capacity of the host population (if all could be used)? Are the many hosts with high average availability? What is the median fraction of time a randomly selected host is available? Answering these questions is helpful e.g. for capacity planning and deciding on which hosts a job is most likely to finish on time.

A related metric is the average duration of host's availability (aka *average availability run*, *aveAvaRun*). Note that for a fixed value of *aveAva* the metric *aveAvaRun* can range widely. If a host changes availability state very frequently, *aveAvaRun* might be small (on the order of few minutes); if a host behaves "stable", the value of *aveAvaRun* can reach months. The knowledge of this metric is particularly important to optimize the intervals between result checkpointing. It also helps to select the appropriate hosts for jobs where a non-availability incurs a high cost (e.g. due to data migration overhead). Figure 1.2 (from [4]) shows the cumulative distribution of this metric for the same study as in Figure 1.1. It shows that the median time a host is available is 8 hours, however there are at least 36% of hosts with average uninterrupted availability of more than 20 hours.

Characterizing hosts by such fundamental metrics is simple and beneficial. However, such metrics do not capture more complex temporal patterns (e.g. weekly periodicity) and have limited value for short-term availability predictions of specific hosts.

1.2.2 Individual availability models

This section discusses several methods for modeling and forecasting the availability of *individual* hosts. After looking at methods for short-term prediction (partially based on models) we turn our attention to characterization of long-term availability behavior.

1.2.2.1 Short-term prediction

Short term availability prediction attempts to forecast availability of a specific host in a time interval called *prediction interval* $[t, t + pil]$. Here t is the prediction time (usually the "current time" in the deployment scenario) and pil the *prediction interval length*. To simplify, we assume that pil is an integer indicating hours and consider a host as available in $[t, t + pil]$ iff it is completely available (i.e. without interruptions) in this time interval. In the following we describe several types of predictors for this scenario.

Last value predictor. The last value predictor (abbreviated `LastVal`) is a simplistic predictor which uses the availability value in the last hourly interval before prediction (i.e. $[t - 1, t]$, where t is in hours) as the prediction of availability for the interval $[t, t+pil]$. Its advantage is virtually non-existing computational and memory cost due to lack of a past availability model.

Gaussian models of availability runs. This algorithm (abbreviated `Gauss`) models the lengths of both availability and non-availability *runs* (i.e. uninterrupted periods of availability or non-availability). To train a model we compute the average and standard deviation of the length of all availability runs in the training interval data (same for non-availability runs). Subsequently the run lengths are modeled by the Normal (Gaussian) distribution. To predict, we first compute the expected remaining length of the current run. This is given by the number k of hours since last availability change and expected (total) run length (derived from the distribution). If the last observed state is availability, we compute the probability p that the end of prediction interval (i.e. $k + pil$) is still in the current availability run. If the last observed state is non-availability, we use the Gaussian model for non-availability runs to find out whether this run is likely to extend to the first hour of the prediction interval (which gives "non-available" as the prediction value). Finally, if the standard deviation of a run length is above a specified threshold (separately for availability and non-availability runs), we revert to the `LastVal` predictor. This ensures that even if the run lengths are non-stationary predictions of a reasonable quality can be achieved.

Classification-based predictors. Classifiers are well-studied algorithms

with broad applicability in data mining. They are typically used when inputs and outputs are discrete [5] as in our case. Formally, a *classifier* is a function $f : V \to W$ which assigns to a vector $v \in V$ a label $w \in W$ where W is a discrete set [6]. The scalars in v correspond to *attributes* - essentially, properties of an object which we would like to label. In the *learning* or *training phase* a classification algorithm is presented a set of examples (v, w) (attribute vectors with correct labels) from which it attempts to build a classifier f which captures the relationship between attribute values and labels. Subsequently, *classification* is performed: a vector v with an unknown label is given, and the most likely label $f(v)$ is computed.

In our setting the label w is 1 or 0 depending whether a host is available in the complete interval $[t, t+pil]$ or not. Attributes are (in the basic version) past availability values of the considered host starting at times $t - k$ (for multiple $k > 0$). Thus, we combine in a training example (v, w) measurements v of past availability with a label w indicating "current"/"future" availability. This effectively turns classification into (availability) prediction. The training examples are obtained from a specific time interval called *training interval*; essentially, each hour of this interval gives rise to one training example.

Since the above-described "raw" attributes might not be a good data representation to extract availability *information* from the past data we have included additional functions of historical data (*derived attributes* or *features*) to extend the attribute vector. These features are:

- *Time*: time and calendar information of the last hour prior to t
- *SwitchAge*: number of hours since last change of availability until t
- *Averages*: averages of the host availability over the last 2, 4, 8, 16, ..., 128 hours before t.

We have studied several classifiers, including Naïve Bayes (abbreviated `NB`) [7], Support Vector Machine SMO (`SMO`) [8], K^*- an instance based classifier (`K*`) [9], multinomial logistic regression model with a ridge estimator (`Logistic`) [10]. Other methods such as a C4.5 decision tree [11] are also possible.

Estimating prediction accuracy. Independently of the method, the prediction results are evaluated on a *test interval*, a data segment following directly a training interval (the training data is ignored for `LastVal`). To estimate the accuracy we use a *prediction error* defined as a ratio of mispredictions to all predictions made on the test interval (in case of our 0/1 data this corresponds to the popular *Mean Squared Error*, *MSE*). As availability patterns of hosts are likely to change over time, we segment data (for a specific hosts) into a series of partially overlapping training intervals and create a new model for each one. The complete prediction error of model-based predictors is computed over a succession of the corresponding test intervals.

aveSwitches	*aveAva*	*aveAvaRun*	*aveNavaRun*	*zipPred*	*MSE*
↓	↑	↑	↓	↓	↓

Table 1.2: Long-term availability metrics and their sort order (↑ = ascending, ↓ = descending) for regularity rank computation

1.2.2.2 Long-term modeling and host ranking

As noted in Section 1.2, analyzing past availability behavior can be beneficial in a multitude of ways. Our focus here is on exploiting the phenomenon that long-term properties of availability of (individual) hosts can influence short-term prediction accuracy. For example, if a host is switched on and off frequently and randomly, the individual prediction models will be significantly worse that for hosts which are rarely switched on / off or these events follow a regular pattern.

To this purpose we assign each host a *regularity rank* $r = 1, \ldots, k$, where 1 corresponds to most "unpredictable" ("irregular") hosts and k to most "predictable" ("regular") ones. This rank is assigned according to one of several long-term availability metrics M of hosts ([12, 13]) as follows. For a specific M we compute its value for each host and sort them by these values (ascending or descending depending on M, see Table 1.2). Subsequently, hosts are subdivided into k equally-sized groups $r(1), \ldots, r(k)$, each corresponding to a single rank value.

This first metric called *aveSwitches* is defined as the number of changes between availability and non-availability (or vice versa) per week. Metric *aveAva* is the average host availability in the training data. The *aveAvaRun* (*aveNavaRun*) is the average duration of uninterrupted availability (non-availability). The metric *zipPred* is more involved and computationally costly. The former is inspired by [14] 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 final metric - *MSE* - we train a cheap predictor on a part P_1 of the training data (P_2 is not used later) and classify the remaining training data P_2. The Mean Squared Error (Section 1.2.2.1) of the classification on P_2 is used as the error score. The latter metric has turned to be most useful for finding the regularity ranks on SETI@home hosts and is used in case studies of Section 1.4.

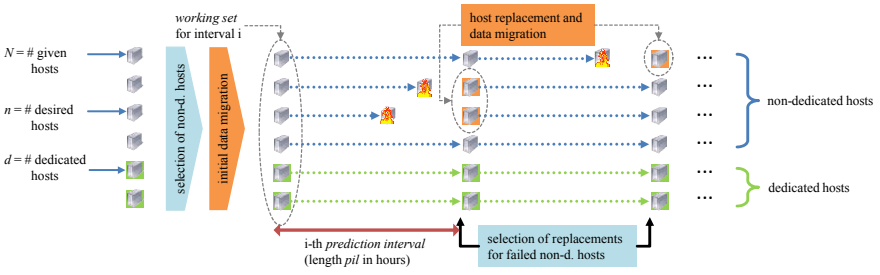

Figure 1.3: Illustration of the resource provisioning process across many prediction intervals

1.3 Collective Availability

As outlined in Section 1.1.2 generally it is not possible to influence the short-term or average availability of a *specific* non-dedicated host[1]. Consequently, the traditional notion of (individual) availability needs to be a relaxed and subsumed by *collective availability* introduced in [12]. The latter is *achieved* if in a pool of N hosts at least n remain available over a specified period of time. If an application to be provisioned requests a set of n *desired hosts*, the "resource owner" assigns to it a *working set* of $N \geq n$ hosts with a specific level of *redundancy* $R = (N - n)/n \geq 1$. Obviously higher redundancy increases the chances that the collective availability is achieved. Besides of adding redundant hosts (i.e. increasing N) it is possible to increase the success rate of achieving collective availability by adding *dedicated* resources (e.g. EC2 instances). The working set is then composed of d dedicated hosts and $N - d$ non-dedicated hosts. This approach and the availability vs. cost trade-offs has been studied in [13].

To ensure uninterrupted operation over a long time period the selection of the working set needs to be repeated periodically as illustrated in Figure 1.3. The time interval for which a working set is selected and operated corresponds to the prediction interval *pil* introduced in Section 1.2.2.1. At the end of each prediction interval hosts which dropped out needs to be replaced by other (non-dedicated) hosts (selected via short-term predictions as shown in Section 1.2.2.1). Obviously each replacement of a failed host might incur some costs due to transfer of working data and host setup. To quantify these costs, we introduce the *migration rate* $mr = m/N$ where m is the average number of (non-dedicated) hosts which fail during a prediction interval (and N the

[1] The exception are Amazon EC2 Spot Instances, where availability can be controlled to a large degree by setting different instance bid prices. This mechanism is used in [15] for cost-efficient provisioning of divisible workloads.

working set size). Note that if collective availability is always achieved mr is never larger than $(N-n)/N$. The selection and assignment of working sets is assumed to be done by a central instance, but there are no inherent constraints to use a fully distributed (Peer-to-Peer) or a hierarchical architecture.

1.3.1 Statistical guarantees and cost vs. migration rate

The notion of collective availability allows us to give (probabilistic) availability assurances. In detail, in a single prediction interval we strive to fulfill an *availability guarantee* y defined as the probability that at least n of the hosts in the (possibly larger) working set remain available over time *pil*. In other words, y is a probability that the collective availability is achieved in a single prediction interval. If there are only non-dedicated resources (i.e. $d = 0$) the only way to keep availability guarantees (for desired n and y) is to adjust the redundancy R (effectively, to change the total working set size N). The minimum sufficient value of R depending on n and y can be derived empirically by simulations on historical data (see Section 1.4); this relation depends on the availability characteristics of considered hosts. Note that by only changing R we have no control on the migration rate.

The possibility to include d dedicated hosts in our working set allows adjusting the migration rate and gives more options how to provide availability guarantees. On the other side, deployment of dedicated hosts incurs additional (monetary) costs. This leads to a cost vs. migration rate trade-off: larger d implies lower migration rate yet higher resource costs. We introduce the following schema to offer the "application operator" a spectrum of options with different cost and migration rate (or reliability) characteristics.

For given n and y we compute via simulation a representative set Z of Pareto-optimal combinations of N and d. Each pair (N, d) fulfills the availability guarantee y and is *Pareto-optimal* in the sense that neither N nor d can be decreased (leaving the other parameter constant) without violating y. Each such pair (N, d) has an associated *total cost tc* (sum of the cost of the dedicated hosts and the optional monetary costs of migration) as well as the migration rate mr which are straightforward to calculate. Subsequently, we sort the pairs (N, d) according to the (increasing) total cost and again according to the (increasing) migration rate. If an application operator has an upper bound on the total costs, to provision the application she chooses a pair (N^*, d^*) with a total cost just below this threshold which minimizes the migration rate. An analogous approach is used if an upper bound on the migration rate is specified. The selected combination (N^*, d^*) is then used in the resource provisioning process over many prediction intervals as illustrated in Figure 1.3.

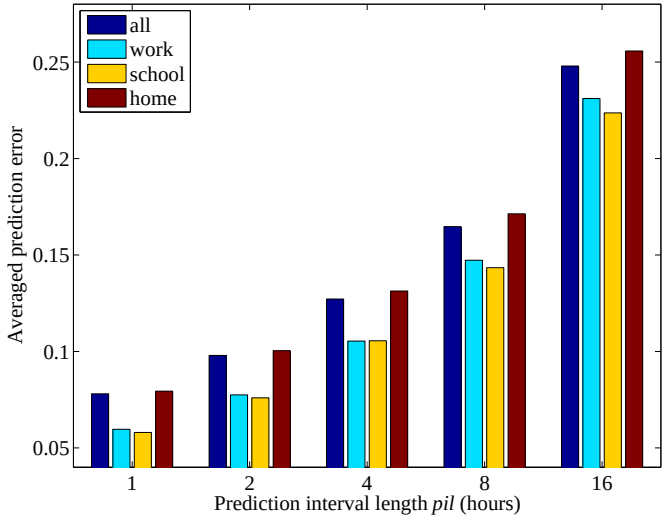

Figure 1.4: Prediction error depending on the host type and *pil*

1.4 Case Studies for SETI@home-Hosts

1.4.1 Availability prediction

To illustrate the techniques from Section 1.2 we show selected results of the availability prediction published in [12] and [13]. They were computed on random subsets of more than 112,000 hosts that were actively participating in the SETI@home project [16] between December 1, 2007 and end of February 12, 2008 (the subset sizes range from 10,000 to about 48,000). 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. We assume that the CPU is either 100% available or 0% which is a good approximation of availability on real platforms. The availability data per host is a string of bits, each representing non-availability (0) or availability (1) in a particular hour.

We first investigate the influence of the host type and prediction interval length (*pil*) on the error (MSE) of the short-term prediction. Here `NB` is used as a representative classification algorithm. Figure 1.4 illustrates that the host type influences consistently the prediction error, with work and school hosts being more predictable. The figure shows also a strong influence of the prediction interval length, *pil*, on the prediction error. This is a consequence of increased uncertainty over longer prediction periods. We have also studied impact of the length of the training data interval on accuracy (not shown). It turns out that for training data sizes of more than 20 days accuracy improve-

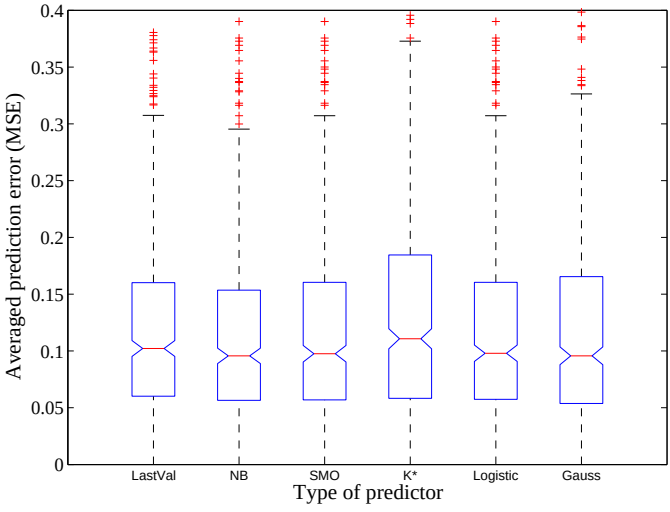

Figure 1.5: Comparison of accuracy of the prediction methods ($pil = 4$)

ments are marginal. Therefore, a training data interval of 30 days is used in all other experiments.

As next we compare the introduced prediction techniques. Figure 1.5 illustrates for $pil = 4$ the MSE of each prediction algorithms averaged over 500 hosts selected randomly from our data set. The boxplot shows the lower quartile, median, and upper quartile values in the box; whiskers extend 1.5 times the interquartile range from the ends of the box while "crosses" outside them are considered outliers. Obviously `NB` and `Gauss` have smallest median prediction errors but all algorithms except for `K*` perform similarly well. For $pil = 1$ highest accuracy was attained by simple predictors like `LastVal` and `NB` but again the differences are not very pronounced (see [13]).

Subsequently, we study how derived attributes influence accuracy of classifier-based predictors. Figure 1.6 shows averages of MSEs (over 500 hosts as above, $pil = 4$) for such predictors and various derived attributes added one at a time to the base input (see Section 1.2.2.1 for explanation of abbreviations). Except for `SMO` and time-based derived attributes, all errors are larger. Similarly, this holds for the case when $pil = 1$ (not shown).

In general, in case of this data sophisticated predictors do not have any consistent advantage over the extremely simplistic approach such as `LastVal`. One explanation is that only feature-scarce data is available for training (essentially this data is just a bit string representing past availability) which is insufficient to create sophisticated models.

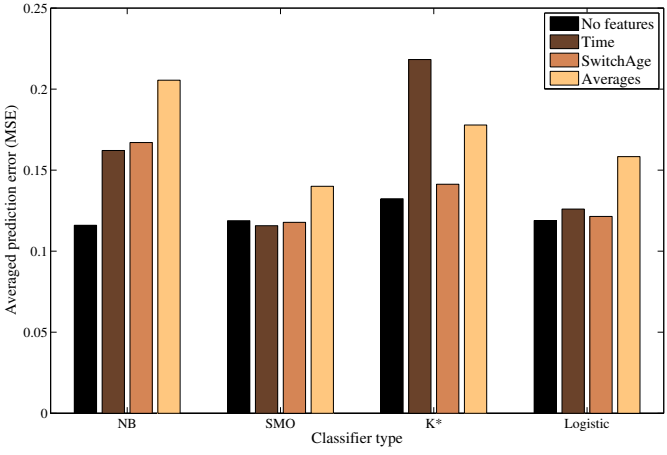

Figure 1.6: Influence of the derived attributes for classifier-based prediction and $pil = 4$

1.4.2 Availability guarantees

In this section we demonstrate how increasing levels of redundancy R are needed to achieve the desired the availability guarantee y, all this depending on the regularity rank r (Section 1.2.2.2). For this study the same data as in Section 1.4.1 is used; see [12] for all details.

To determine the minimum necessary redundancy level R for the desired availability guarantee y we change R and in each case observe via trace-based simulations the *success rate*. The latter metric is the fraction of trials (at a fixed $R = R^*$) for which collective availability has been achieved. Interpreting the success rate as the probability of achieving the collective availability for redundancy level R^* we can find a minimum R which satisfies the desired availability guarantee y by a simple table look-up.

We each trial we randomly choose N number of hosts from the pool predicted to be available for the entire prediction interval (with $pil = 4$ hours). We run trials in this way throughout the test period of two weeks. For each data point shown in the figures, we ran about 30,000 trials to ensure the statistical confidence of our results. We also consider two regularity ranks $r = 1$, 2 derived from the metrics *aveSwitches*. Figure 1.7 shows our findings for working set sizes N of 1, 4, 16, 64, 256 and 1024. As expected, for the lower regularity rank $r = 1$ much higher redundancy levels are needed to achieve the same success rate as for $r = 2$. For example, with $r = 2$, a redundancy of 0.35 will achieve success rates of 0.95 or higher. With $r = 1$, 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 ranking hosts by regularity levels (here using the *aveSwitches* indicator) significantly

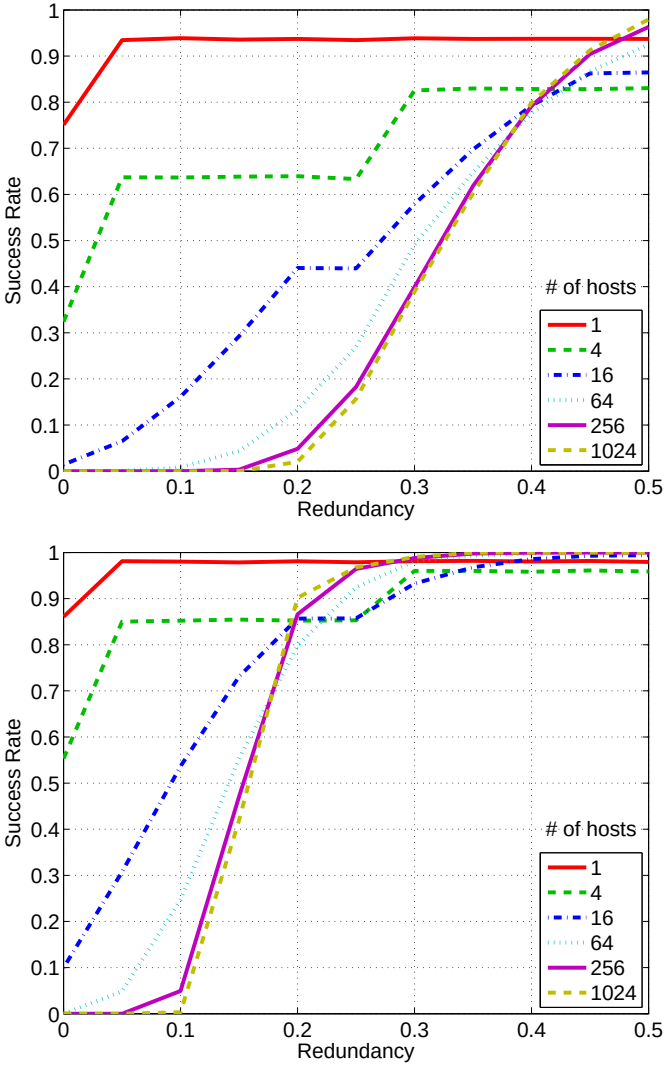

Figure 1.7: Success rate depending on redundancy R for hosts with regularity rank $r = 1$ (top; low predictability) and regularity rank $r = 2$ (bottom; high predictability) for $pil = 4$

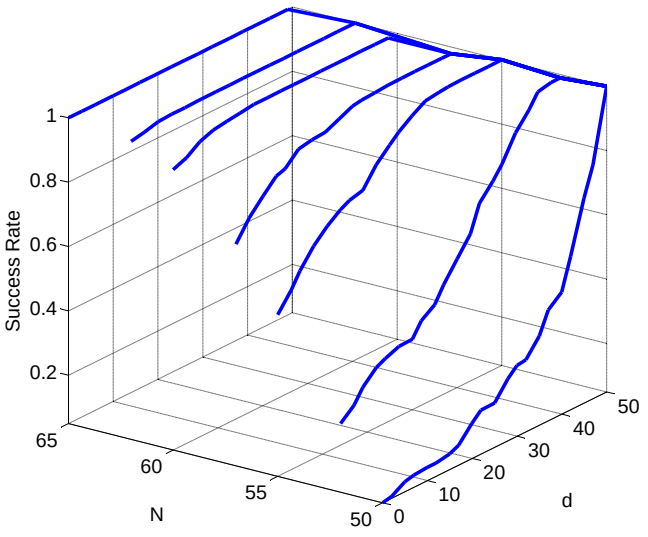

Figure 1.8: Success rate depending on the total size N of the working set and number d of dedicated hosts ($n = 50$, $pil = 4$, rank 4 of 4)

improves the accuracy of short-term availability prediction, and consequently the efficiency of achieving collective availability.

1.4.3 Cost vs. migration rate

As the last showcase of experimental results we consider mixtures of SETI@home hosts and dedicated resources (with costs modeled after Amazon EC2 instances) and study the trade-offs between costs and higher migration rates discussed in Section 1.3.1. To this end we execute trace-driven simulations of the scenario from Figure 1.3 over the entire trace period excluding the training interval.

We consider different *pil*'s of either 1 or 4 hours and independently also conduct different experiments for 4 regularity ranks $r(1)$ to $r(4)$ obtained by the MSE metric (Section 1.2.2.2). We also consider different numbers n of requested hosts (50 or 100). For each n, we further vary the working set size N such that the redundancy R varies from 1 (none) to 1.6. For each N, we vary the number d of dedicated hosts allocated from 0 to n. Each such simulation is repeated 4000 ($pil = 1$) or 2400 ($pil = 4$) times with varying starting point to ensure statistical confidence.

For each simulation several metrics are collected. The first one is the success rate defined in Section 1.4.1 (which corresponds to the availability guarantee y for each combination (N, d)). The second metric is the total cost of achieving this service level. Here as the host cost we assume 10 US cents per

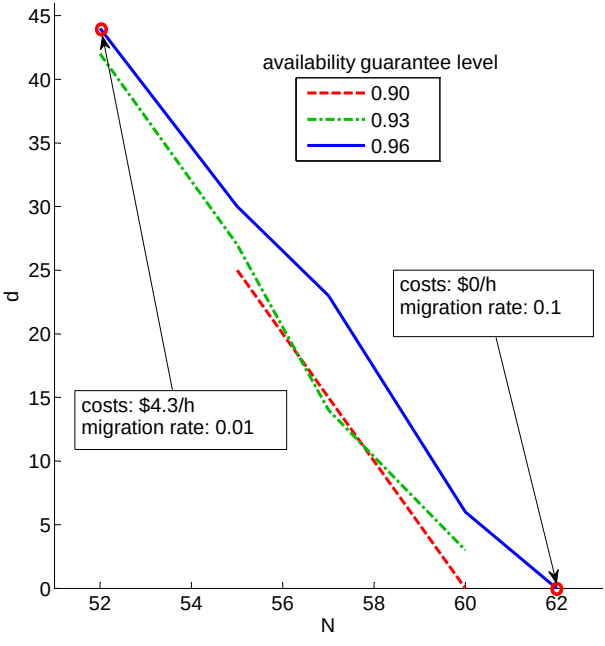

Figure 1.9: Pareto-optimal combinations of N and d for $y = 0.9$,0.93 and 0.96 ($n = 50$, $pil = 4$, rank 4 of 4)

hour, which is equivalent to the hourly rate for a small instance on Amazon's EC2. The third metric is the migration rate needed to achieve the service level.

Figure 1.8 shows the first metric (success rate) depending on the total size N of the working set and number d of dedicated hosts for $n = 50$, $pil = 4$ and rank 4 (other cases are shown in [13] with rank order "reversed"). From these result we obtain a set Z of Pareto-optimal combinations of N and d by "intersecting" the surface shown in Figure 1.8 with a plane parallel to x-y axis at a desired value y of the success rate. Each pair (N, d) in Z obviously fulfills the availability guarantee y (and is Pareto-optimal in respect to N or d).

Figure 1.9 shows these sets for three levels of availability guarantees y of 0.90, 0.93 and 0.96. By computing the total cost and the migration rate for each pair (N, d) in Z we can investigate the trade-off between these two metrics. Figure 1.9 illustrates this trade-off for $y = 0.96$ at the extreme ends of the solution spectrum: while using no dedicated resources eliminates costs, the migration rate is 0.1 (and working set size N becomes 62). On the other hand, using many dedicated resources ($d = 43$) incurs significant costs (4.3 USD per hour) but lowers the migration rate to 0.01. This effect allows to select an appropriate (usually application-specific) combination of dedicated

and shared hosts in order to optimize monetary costs or the migration rate or both.

1.5 Conclusions

Lack of control on availability on individual hosts is one of the major obstacles for effective usage of non-dedicated resources. In this chapter we have outlined some approaches to overcome this problem. These approaches combine availability modeling and schema for masking outages of individual hosts via redundancy. Availability modeling comprises distribution models as well as short-term and long-term characterization of individual hosts. The major benefit of such models is their ability to identify hosts which are likely to be available within next few hours or which behave "nicely" in respect to availability on a longer-term time scale. As a method for masking outages we have introduced collective availability together with probabilistic guarantees. To achieving collective availability at a minimal monetary and resource we have investigated the relation between redundancy, probabilistic guarantees and the migration rate (host replacement rate) between "provisioning epochs". Besides redundancy, mixing-in dedicated hosts is helpful to increase collective availability level at a low host replacement rate yet it implies higher monetary costs.

The proposed mechanisms assume applications which perform efficiently even in face of frequent failures of a significant part of resources (such as MapReduce). Adapting applications to these challenging deployment conditions is a non-trivial problem which is likely to (and should) drive a part of the activities within the researchers community devoted to non-dedicated resources.

Bibliography

[1] W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, G. Gedye, and 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.

[2] J. Bezos, "Amazon.com: Amazon EC2, Amazon Elastic Compute Cloud, Virtual Grid Computing: Amazon Web Services," http://www.amazon.com/gp/browse.html?node=201590011.

[3] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. Morgan Kaufmann, San Francisco, 2005.

[4] D. Kondo, A. Andrzejak, and D. P. Anderson, "On correlated availability in internet-distributed systems," in *9th IEEE/ACM International Conference on Grid Computing (Grid 2008)*, Tsukuba, Japan, September 29–October 1 2008.

[5] R. Vilalta, C. V. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss, "Predictive algorithms in the management of computer systems," *IBM Systems Journal*, vol. 41, no. 3, pp. 461–474, 2002. [Online]. Available: http://www.research.ibm.com/journal/sj/413/vilalta.html

[6] T. Hastie, R. Tibshirani, and J. Friedman, *Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd Ed.)*. New York: Springer Verlag, 2009.

[7] G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proc. 11th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1995, pp. 338–345.

[8] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 185–208.

[9] J. G. Cleary and L. E. Trigg, "K*: an instance-based learner using an entropic distance measure," in *Proc. 12th International Conference on Machine Learning*. Morgan Kaufmann, 1995, pp. 108–114.

[10] S. le Cessie and J. C. van Houwelingen, "Ridge estimators in logistic regression," *Applied Statistics*, vol. 41, no. 1, pp. 191–201, 1992.

[11] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[12] A. Andrzejak, D. Kondo, and D. P. Anderson, "Ensuring collective availability in volatile resource pools via forecasting," in *DSOM*, September 2008, pp. 149–161.

[13] ——, "Exploiting non-dedicated resources for cloud computing," in *12th IEEE/IFIP Network Operations & Management Symposium (NOMS 2010)*, Osaka, Japan, Apr 19–23 2010.

[14] E. J. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards parameter-free data mining," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2004, pp. 206–215. [Online]. Available: http://doi.acm.org/10.1145/1014052.1014077

[15] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under SLA constraints," in *18th MASCOTS*, Miami Beach, August 17–19 2010, pp. 257–266.

[16] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *5th International Workshop on Grid Computing (GRID 2004)*, R. Buyya, Ed. IEEE Computer Society, November 4 2004, pp. 4–10.