Failure Prediction in Computational Grids[1]

Woochul Kang and Andrew Grimshaw
University of Virginia
{wk5f, grimshaw}@cs.virginia.edu

Abstract

Accurate failure prediction in Grids is critical for reasoning about QoS guarantees such as job completion time and availability. Statistical methods can be used but they suffer from the fact that they are based on assumptions, such as time-homogeneity, that are often not true. In particular, periodic failures are not modeled well by statistical methods. In this paper, we present an alternative mechanism for failure prediction in which periodic failures are first determined and then filtered from the failure list. The remaining failures are then used in a traditional statistical method. We show that the use of pre-filtering leads to an order of magnitude better predictions.

1. Introduction

Ensuring QoS properties such as job completion time or availability in grids often requires predicting whether a particular resource such as a host will be functioning over some time interval [21, 23, 24, 5]. For example, if we wish to start at time *T* a job *J* that takes 8 CPU hours and guarantee that it will complete at some time *T+8* on a host *H* with probability *P*, then we need to be able to determine the probability that the host will continue to function over that interval. If we cannot find some host that meets these requirements, then we may need to search for two or more hosts so that we can run multiple copies of *J* and know that, with probability *P*, at least one copy of *J* will complete. The problem is how do we estimate the probabilities.

More formally, let Δt be a time interval with a start time and a stop time, and let $P_i(\Delta t)$ be the probability that host H_i will function over the interval Δt. How do we compute the P_i's? One obvious solution is to use statistical methods based on historical data of the up and down states of the machines. The problem, as we'll see in Section 4, is that statistical models perform very poorly in environments where there are a significant number of periodic events such as when machines are automatically rebooted on some schedule, when some machines are turned off each night, or when weekly backups take machines off-line. Indeed, in shared workstation environments, if we model a host as being unavailable or "down" when a user is at the keyboard, one can observe regular patterns of availability.

To address the shortcomings of traditional statistic techniques in failure prediction in large-scale grids composed on resources controlled by others, we have developed a *filtered failure prediction model (FFP).* The basic idea of FFP is simple - we assume that we have an event history for each host that consists of a series of UP/DOWN events. We first determine periodic events, note them, filter them out, and then pass the remaining events onto a traditional statistical method. Then, to compute $P_i(\Delta t)$, we first check against periodic failures, and if not affected by a periodic failure, use the statistical method to determine the probability of success. We have found that FFP significantly outperforms time-homogeneous statistical techniques – allowing us to provide more accurate QoS bounds.

The remainder of this paper is as follows. We begin with a thorough examination of statistical techniques and why they do not work well in an environment with periodic events. We then present FFP in detail, including the results using FFP versus a standard method on a large number of machines at the

[1] Partially funded by NSF-SCI-0426972.

University of Virginia. We conclude with a summary and discussion of future work.

2. Related Work

There has been a great deal of work on analyzing events and building models for machine/resource behavior.

Some have focused on analyzing error and failure event logs [16, 18, 24]. The work of Lin et al.[18] is similar to our approach in the sense that they see the failures and errors due to multiple sources, not a single source. They separate sources of errors into intermittent and transient, and provide a set of rules for fault prediction. They show that the time between errors from each separate source follows a Weibull distribution and the combined errors cannot be modeled by any well-known distributions. However, their approach uses information obtained in a controlled environment. In Grids, we have very limited control on resources and the available information about resource error and failure. Therefore, the approach of Lin et al. cannot be used for our research as it stands. Our approach does not rely on detailed information from a resource. In the extreme case, the only information that we can get is ping results to check the liveness of a resource.

Sahoor et al.[24] discussed critical event prediction in terms of proactive management in autonomic computing. This study does critical event prediction in large-scale computer clusters. They suggest the use of linear time-series models and rule-based classification techniques to do rare event prediction. The prediction is made by detecting occurrences of a set of event types in a time window. This approach also assumes that a predictor has detailed information about event types, which is rarely available in Grid.

The preprocessing of the original data set of events to simplify analysis has been done before, e.g., [16] and [17]. However, the techniques have been mostly used to eliminate redundant information such as successive repeated errors. In our case, the original series of events are preprocessed to separate periodic events from non-periodic events.

The concept that a resource's lifetime follows some kind of statistical distribution has existed for a long time and has its basis in queuing theory. Some [6, 13] have used processes/machine lifetime distribution for load balancing. In [13], the process lifetime distribution is used to initiate dynamic migration. Typically, they all assume some probability distributions for the process lifetime. However, the distribution is about the length of process lifetime when it finishes normally. They do not include the termination of a job from errors and resource failure.

The emergence of Grid as a new distributed computing platform has fostered many studies on resource availability/reliability prediction in Grid [5, 23, 4, 27]. Brevik et al.[5] compare parametric and non-parametric methods for predicting machine availability. Their result shows that a non-parametric approach is better in most experiments in estimating the lower bound of a given quantile, especially when the sample size is small. A non-parametric approach is very useful when the distribution is not easy to be fitted by parametric methods as in our case. However, the problem with non-parametric approaches is that they are geared toward testing assertions rather than estimation of effects. From a practical viewpoint, a job scheduler needs to estimate and quantify the reliability at some specific time to test and compare the fitness of the resources. Our work shows that parametric approaches are problematic in the presence of periodic failures, but the distribution can be fitted using parametric methods after filtering of periodic failures. Ren et al.[23] used a semi-Markov model to make a long-term prediction of machine availability. A semi-Markov model can make more accurate predictions than an ordinary Markov model with the expense of much higher computational complexity [20]. Our work complements their approach by filtering periodic unavailability before the modeling and prediction of non-periodic events.

Despite the large body of work on error/failure prediction, the absence of any research on periodic resource unavailability as a separate source of failure and modeling of them to make a prediction in job scheduling is the motivation of our study in this paper.

3. Input Data

To understand the behavior of a large collection of machines, we set up a monitoring environment at the University of Virginia [14] that collected up/down information at 5-minute intervals for over 700 machines over a three-month period. We observed that the machines fell into two broad classes – managed machines that experienced regular rebooting and those that did not. Rebooting is an instance of the idea of "software rejuvenation" [15]. The idea is to periodically restart software to prevent errors from accumulating.

Figure 1 below shows the inter-event time distribution of failures for a single "managed"

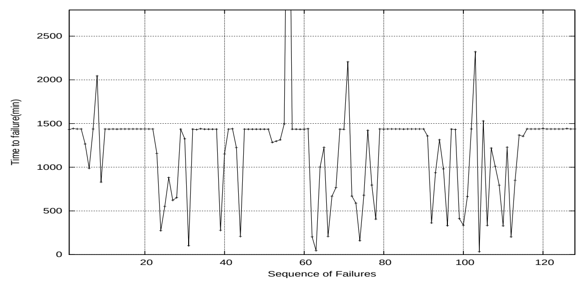


Figure 1. Inter-arrival intervals of failures on a “managed” machine in minutes

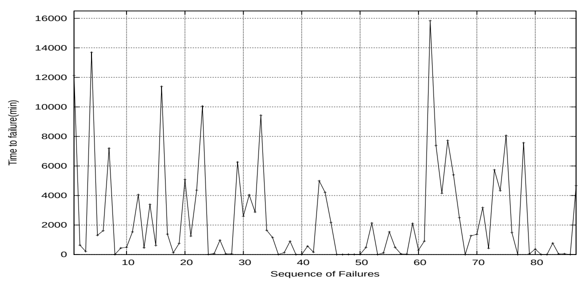


Figure 2. Inter-arrival intervals of failures on a typical “unmanaged” machine in minutes

machine. Each point represents a down-time. The height of the point indicates the number of minutes of uptime that preceded the failure. Note the large number of reboots at approximately 1440 minutes (one day). You can also see that occasionally the machine missed a reboot – and that scheduled reboots are not the only cause of reboot. Because of the reporting interval, the machine may appear to have missed a reboot. That is not the case – rather, it rebooted quickly enough to meet its next required report-in. In the case of unmanaged machines, the story is different. As shown in Figure 2, there is no clear pattern.

We will use this data set throughout the remainder of the paper.

4. Standard Statistical Methods

Given the problem of predicting uptimes, one would normally think to use simple statistical methods such as fitting the collected data to a continuous model. The exponential distribution [25] model is most common and easiest to use for failure analysis. For a more exact estimate, a Weibull distribution [25] is commonly used. Optimization-based model fitting techniques such as MLE (Maximum Likelihood Estimator) can be used to get two parameters, shaper parameter and scale parameter, of a Weibull distribution to fit the empirical data [25].

For a resource which does not show periodic failure, optimization-based model fitting techniques work reasonably well if we choose the right model. For example, Figure 3 shows that time-to-failure (TTF)
distribution of an unmanaged machine can be approximated by Weibull distribution. The fitted exponential distribution shows a bigger gap than the Weibull from the empirical distribution, but it can be used when the exact estimate is not required.

In contrast, the distribution of resource with periodic unavailability cannot be easily modeled by any single well-known statistical distribution because it does not resemble any one of them. In Figure 4, the distribution of the managed machine shows sudden discontinuity at 1440 minutes. This discontinuity comes from the fact that the machine reboots every 24 hours. The empirical distribution obtained from a managed machine cannot be easily approximated by

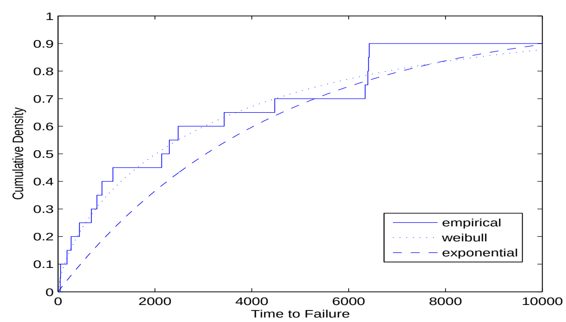


Figure 3. Time-to-fail distribution of an unmanaged machine and its fitting to Weibull and exponential distribution

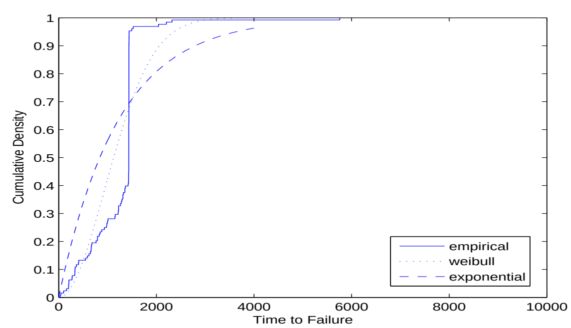


Figure 4. TTF distribution of a managed machine and its fitting to Weibull and exponential distribution

any statistical distribution. And, as a consequence, the prediction from predictors that do not consider these kinds of periodic resource unavailability can be either

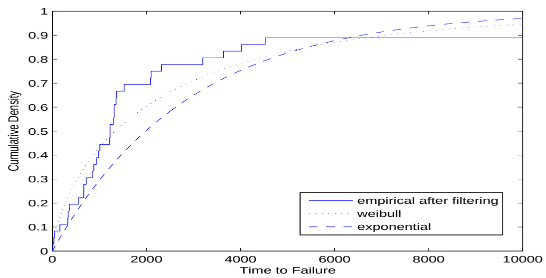


Figure 5. Time-to-fail distribution of a managed machine after filtering out periodic failures and its fitting to Weibull and exponential distribution

inaccurate or totally wrong. For example, the empirical distribution shows us that the probability of a job to live longer than 24 hours is almost 0. However, the fitted models tell us that there is more than a 30% chance of survivability after 24 hours, which is totally misleading.

5. Filtered Failure Prediction

In the previous section, we showed that we can model failures of a resource using ordinary statistical methods but that the models perform poorly in the presence of periodic events. In this section, we present a Filtered Failure Prediction technique where failures of a resource are categorized into periodic and non-periodic to make a more effective prediction.

5.1. Filtered Failure Prediction Model

Figure 5 shows what happens if periodic failures are filtered out from the original signal. The same up/down signal used to generate Figure 4 is used but periodic failures are removed from the signal. After this filtering, optimization-based fitting techniques are

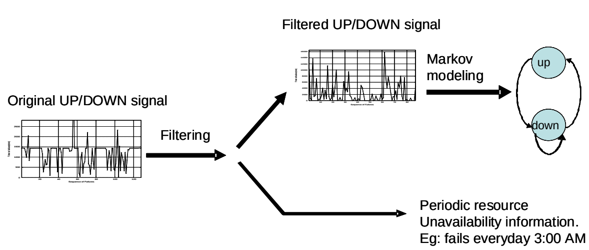


Figure 6. The Markov model is built from filtered machine up/down signal. The information about periodic resource unavailability is also obtained from the filtering

used again to fit the empirical time-to-fail distribution to a Weibull and an exponential distribution. The discontinuity observed in Figure 4 is gone and the empirical distribution is more closely approximated both by Weibull and exponential distribution with smaller gaps. Therefore, if we assume there will be no periodic failure in the future, we can more precisely predict the reliability of a resource with approximated statistical models.

This motivates us to propose *Filtered Failure Prediction (FFP)* where periodic and non-periodic failures are identified and treated as separate independent sources.

In FFP, a sampled signal of resource up/down is fed into a filter which detects and separates periodic failure signals from the original. The filtered up/down signal is used to build a model for reliability prediction. Here, we use the Markov model for its simplicity with reasonable precision. The Markov model built from a filtered signal is called the filtered-Markov model hereafter. If the sampled resource up/down signal has a periodic component, it is detected and used to predict the next time when the resource is unavailable due to periodic failure.

Keep in mind that our objective is to predict $P_i(\Delta t)$ and that Δt includes both a start time and a duration. The reliability acquired from the filtered Markov model is only valid when the expected execution time of a job does not span the next periodic failure point. The reliability from FFP is the conditional probability that the job's execution does not span the periodic failure point. Therefore, the final predicted reliability can be summarized as follows:

$$\text{reliability}_{FFP}(\Delta t)$$
$$= \text{Probability(no failure over } \Delta t \mid \Delta t \text{ does not span the next periodic failure point)}$$
$$= \alpha \cdot \text{reliability}_{\text{prediction from filtered Markov model}}(\Delta t)$$
, Where
$$\alpha = \begin{cases} 1 & \text{if } \Delta t \text{ does not spans the next periodic failure point} \\ 0 & \text{otherwise} \end{cases}$$

For example, assume a resource has periodic failures every 24 hours, its predicted reliability from the filtered Markov model is 90% after 10 hours, and its last periodic failure occurred at time *t*. At *t+5* and *t+15*, the resource is requested to execute a job which needs 10 hours CPU time and more than 80% reliability. The first request can be accepted because the resource is more reliable than is required (90% > 80%). However, the second request cannot be accepted because its run spans the next periodic failure

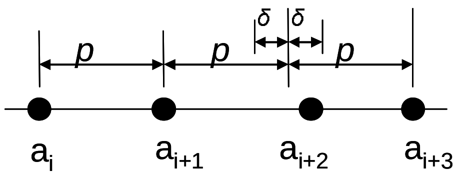


Figure 7. The sequence of "down" events. p is period and δ is time tolerance

point. Its reliability after 10 hours from *t+10* is 0% (= 0*90%), not 90%.

5.2. Filtering of regular events

The most common approach to finding periodicities is the *fast Fourier transform*. The fast Fourier transform provides the means of transforming a signal defined in the time domain into one defined in the frequency domain. However, using fast Fourier transform is inappropriate in our problem because the time information obtained from Grid is often imprecise. In Grid, geographical long distance between the resources and an entity that monitors resources can lead to imprecise time information, phase shift in periods. Therefore, instead of using fast Fourier transform, we used a modified version of the periodicity detection algorithm from the work of Ma et al.[19].

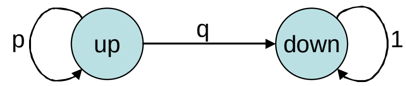
The algorithm takes as input the sequence of down events $\boldsymbol{S=\{a_1, a_2, \ldots, a_N\}}$. If the number of occurrence of periods, $\boldsymbol{a_{i+1.time} - a_{i.time}}$, is bigger than the threshold, $\boldsymbol{C_\tau}$, the period is recognized as periodic. The algorithm uses chi-squared test to determine a 95% confidence level threshold. The number of period occurrences is adjusted to accommodate the time tolerance, $\boldsymbol{\delta}$, of period length. After the lengths of periods are identified, we determine the wall clock time of the next periodic down event as follows:

For each period $\boldsymbol{p_k}$ identified above:

1. Get first event $\boldsymbol{a_i}$ from $\boldsymbol{S}$ such that $\boldsymbol{p + \delta \leq a_{i+1.time} - a_{i.time}}, \leq \boldsymbol{p + \delta}$
2. Count the number of event $\boldsymbol{a_j}$ such that $\boldsymbol{a_{j.time}} = \boldsymbol{a_{i.time} + n \cdot p \pm \delta}$ (n=1,2,3…)
3. If the total count from step 2 is over the threshold, $\boldsymbol{C_\tau}$, from the periodic detection algorithm, output ($\boldsymbol{a_{i.time}}$, $\boldsymbol{p_k}$) and remove $\boldsymbol{a_i}$ of step 1 and $\boldsymbol{a_j}$ s of step 3 from $\boldsymbol{S}$
4. Else get next $\boldsymbol{a_i}$ for step 1, and repeat 1~4

The wall clock times $\boldsymbol{a_{i.time}}$ s as the output of step 3 are the basis times from which we can calculate the next expected periodic down time. The next expected periodic down time for a periodic event type ($\boldsymbol{a_{i.time}}$, $\boldsymbol{p_k}$) is:

Next expected periodic down time =
min($\boldsymbol{a_{i.time} + n \cdot p_k}$ ***) > current wall clock time***, where n=1,2,3…



$$P = \begin{bmatrix} p & q \\ 0 & 1 \end{bmatrix}$$

Figure 8. State transition diagram and its probability matrix

5.3. Modeling of non-periodic availability

Once the failure history has been filtered and periodic events are removed, a model is built from the remaining non-periodic events. For our research, a simple Markov chain as in Figure 8 is used for modeling and prediction.

The reliability of a resource is calculated as the following equation if a client requests to execute a job on it.

$$\text{reliability}_{\text{prediction from filtered Markov model}}(h) = (P^h)_{up,up}$$

The monitoring interval should be chosen to balance between the modeling accuracy and the computational cost. While we may have a more accurate model with a shorter monitoring interval, the cost of monitoring and computation increase. In our experiment, a 5-minute interval is used to monitor the resources and the chain has only two states; up and down.

The assumption behind the Markov chain is that sojourn time in each state follows exponential distribution and the parametric modeling in the previous section shows that exponential distribution is less accurate than Weibull. However, we assume exponential distribution because of its analytical tractability. The shortcomings of the exponential distribution, such as memory-less property, is complemented using periodicity information which tells the wall-clock time of the next periodic failure. If more accurate modeling is required, then FFP can be combined with a semi-Markov model which has a different state sojourn time distribution such as Weibull or hypo-exponential with additional computational cost and complexity [23].

6. Experiment and Results

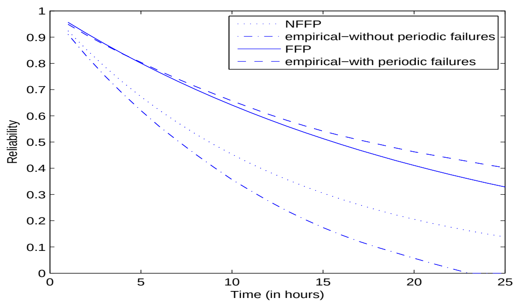


Figure 9. Empirical reliabilities and predicted reliabilities of FFP and NFFP over time

In this section, we compare the prediction accuracy of FFP and traditional techniques and analyze the impact of periodicity on statistical modeling.

6.1. Experiment on prediction accuracy

To investigate the effectiveness of the FFP described in the previous section, we compare the predictive accuracy of an FFP to the ordinary Markov model which does not detect and filter periodic failures. We call this ordinary Markov model a non-filtered failure prediction (NFFP) hereafter. A representative machine was chosen from the machines described in Section 3, and the trace of 3 months from the machine is divided into two parts, one month for training and the remaining two months for validation. Empirical reliabilities are calculated by observing the 2-month validation data. In the case of FFP, empirical reliability is calculated from the same 2-month validation data but from which periodic failures are removed. The predicted and empirical reliabilities as a function of time are shown in Figure 9. The upper two lines are the predictions from FFP and the empirical reliabilities from a trace from which periodic failures are removed, and the two lines below are the predictions from NFFP and the empirical reliabilities from an original trace respectively.

While the prediction from NFFP tells the probability that no failures will happen during a given time interval without consideration of periodic failures, the prediction from FFP is the conditional probability that no failures will happen if the time window does not span the next periodic failure point.

The gaps between the empirical reliabilities and predicted reliability from each model, NFFP and FFP, are the prediction error that indicates their prediction inaccuracy. As expected, FFP is much more accurate than the ordinary Markov model. For instance, the prediction error at 15 hours in NFFP is more than 10% compared to less than 3% in FFP. Moreover, this gap widens rapidly as the time increases in the ordinary Markov model. In contrast, the increase of the gap is much slower in FFP.

The large prediction error from NFFP is problematic in job scheduling. For instance, when we include periodic failures, the empirical reliability after 24 hours is almost 0. However, the predicted reliability from NFFP is about 18%. The scheduler may decide to give high redundancy to increase the reliability because the probability that at least one of them survives after 15 hours is more than 90%. However, this decision is wrong if we are aware of the fact that all computing resources are going to fail at some specific period, particularly in 24 hours in this experiment. In the case of FFP, the scheduler can make a more accurate prediction because it knows the time when a resource is going to periodically fail and the fact that the statistical reliability only has meaning when the expected run does not span over the next periodic failure point.

This prediction accuracy difference between FFP and the ordinary Markov model is not particular to one machine but can be found over all machines in the experiment. The average and 95% confidence intervals of a relative prediction error from 519 machines which have periodic failures are shown in Figure 10. The relative prediction error is defined as follows:

$$prediction\_error_{relative} = \frac{\left| reliability_{empirical} - reliability_{predicted} \right|}{reliability_{predicted}}$$

Figure 10 is the relative prediction errors from NFFP and FFP respectively. We again see that the relative prediction error of an unfiltered model is significant compared to the filtered model. The confidence interval is very small because all machines used in the experiment are homogeneous and under the same management policy. The confidence interval is likely to be larger in other environments.

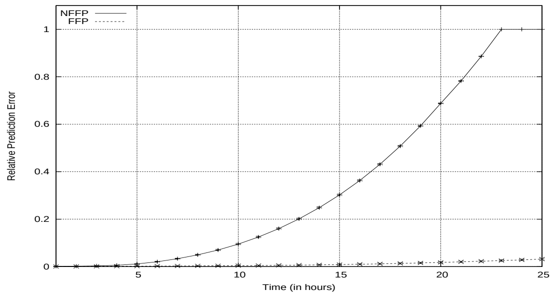


Figure 10. Relative prediction errors with 95% confidence intervals

6.2. Analysis

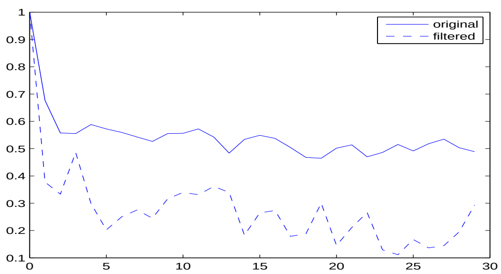


Figure 11. Autocorrelations of filtered and unfiltered time-to-fail series

To understand the effect of periodic resource unavailability on predictability, we did the same analysis as in [27].

We first plot the autocorrelations of filtered and unfiltered time-to-fail series from a managed machine in Figure 11.

The autocorrelation as a function of previous lags reveals how time-to-fail changes over time. The autocorrelation of an unfiltered series is much higher than the filtered series and this implies that the time-to-fail changes relatively slowly. This slow decay in the autocorrelation can be useful for models as the linear time series model in [8] which uses the relationship between current and previous lags. However, most other statistical models including the Markov model relies on the fact that a current state and next state have very limited dependence. Therefore, we believe that the high autocorrelation spoils the predictability of models which assumes limited dependence between lags. We do not consider using linear time series models because even though the model shows good performance in short-term prediction, its prediction accuracy deteriorates rapidly as the length of time window to predict increases [23]. For reliability prediction, long-term prediction, for example reliability after 24 hours, is very typical, and this makes a linear time series model less suitable.

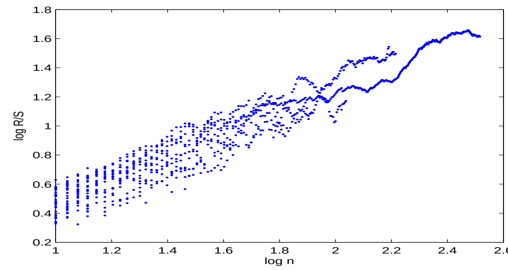


Figure 12. Self-similarity of unfiltered time-to-fail series

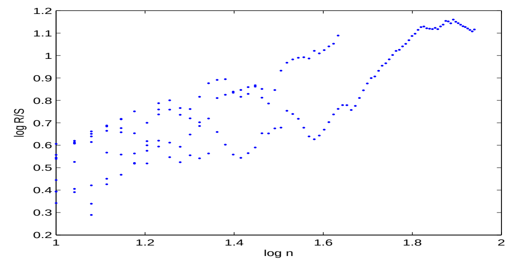


Figure 13. Self-similarity of filtered time-to-fail series

We can also think predictability in terms of self-similarity. As stated in [27], high autocorrelation often implies self-similarity which often is the manifestation of an unpredictable chaotic series.

Figures 12 and 13 shows pox plot [17] for unfiltered and filtered time-to-fail series respectively. The respective Hurst parameters from the pox plots are 0.79 and 0.66. A series is more self-similar if its Hurst parameter is closer to 1. The higher Hurst parameter of the original time-to-fail series without filtering suggests to us that the series can be more chaotic than the filtered. One interesting characteristic of chaotic series is that, even though the series looks similar to random series, a pattern exists in the series. However, a series with high self-similarity seems more chaotic at first glance. As a consequence, the series is less predictable in practical viewpoint [22]. On self-similarity, see [7, 9, 10, 11, 17, 26, 2].

7. Summary and Future Work

Accurate prediction of resource availability is critical to providing completion time, availability, and other quality of service guarantees. Accurate prediction using statistical methods is difficult in the presence of periodic events. While periodic, some resources may claim a particular behavior – it may not always be prudent to trust the information.

We have developed FFP – the Filtered Failure Prediction method that extends traditional statistical methods by first filtering out periodic events. FFP outperforms both exponential and Weibull distributions by more than a factor of 10 in predicting whether a resource will be available for a particular interval. The improvements are particularly pronounced as the time interval approaches the periodic failure interval.

We next plan to integrate FFP into the Genesis II [1] grid system being developed at the University of Virginia. FFP will be integrated into the Genesis II implementation of the OGSA Basic Execution Services (BES)[12] activity factory. BES activity factories take activity documents as parameters. Each activity document contains a JDSL [3] document that describes the job (including optionally the amount of time the job will consume). Activity documents may also contain other sub-documents as extensibility elements. We will extend the BES activity document to contain a dependability document that specifies both the "price" the user is willing to pay as well as the reliability they require. FFP will allow us to make those guarantees.

8. References

[1] http://vcgr.cs.virginia.edu/genesisII/, Genesis II project homepage, 2006.

[2] A. Adas and A. Mukherjee, On resource management and QoS guarantees for long range dependent traffic, INFOCOM '95: Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communication Societies (Vol. 2)-Volume, IEEE Computer Society, Washington, DC, USA, 1995, pp. 779.

[3] A. Anjomshoaa, F. Brisard, M. Dresher, D. Fellows, A. Ly, S. McGough, D. Pulsipher and A. Savva, Job Submission Description Language(JSDL) Specification, Version 1.0, Open Grid Forum, 2005.

[4] W. J. Bolosky, J. R. Douceur, D. Ely and M. Theimer, Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs, SIGMETRICS, 2000.

[5] J. Brevik, D. Nurmi and R. Wolski, Automatic methods for predicting machine availability in desktop Grid and peer-to-peer systems, CCGrid, IEEE, 2004, pp. 190-199.

[6] R. M. Bryant and R. A. Finkel, A stable distributed scheduling algorithm, 2nd International Conference on Distributed Computing Systems, 1981, pp. 314-323.

[7] M. Crovella and A. Bestavros, Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes, Proceedings of SIGMETRICS'96: The ACM International Conference on Measurement and Modeling of Computer Systems., Philadelphia, Pennsylvania, 1996.

[8] P. A. Dinda and D. R. O'Hallaron, An Evaluation of Linear Models for Host Load Prediction, HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society, Washington, DC, USA, 1999, pp. 10.

[9] M. W. Garrett and W. Willinger, Analysis, Modeling and Generation of Self-Similar VBR Video Traffic, SIGCOMM, 1994, pp. 269-280.

[10] M. E. Gomez and V. Santonja, Analysis of Self-Similarity in I/O Workload Using Structural Modeling, MASCOTS '99: Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE Computer Society, Washington, DC, USA, 1999, pp. 234.

[11] S. D. Gribble, G. S. Manku, D. S. Roselli, E. A. Brewer, T. J. Gibson and E. L. Miller, Self-Similarity in File Systems, Measurement and Modeling of Computer Systems, 1998, pp. 141-150.

[12] A. Grimshaw, S. Newhousr, D. Pulsipher and M. Morgan, OGSA Basic Execution Service Version 1.0, Open Grid Forum, 2006.

[13] M. Harchol-Balter and A. B. Downey, Exploiting Process Lifetime Distributions for Dynamic Load Balancing, ACM Transactions on Computer Systems, 15 (1997), pp. 253-285.

[14] H. Huang, J. F. Karpovic and A. S. Grimshaw, A Feasibility Study of a New Mass Storage System for Large Organizations, Computer Science Technical Report, University of Virginia, CS-2005-21 (2005).

[15] Y. Huang, C. Kintala, N. Kolettis and N. D. Fulton, Software Rejuvenation: Analysis, Module and Applications, 25th Symposium on Fault Tolerant Computing, FTCS-25, IEEE, Pasadena, California, 1995, pp. 381–390.

[16] I. Lee, R. K. Iyer and D. Tang, Error/Failure Analysis Using Event Logs from Fault Tolerant Systems, Proceedings 21st Intl. Symposium on Fault-Tolerant Computing, 1991, pp. 10-17.

[17] W. E. Leland, M. S. Taqq, W. Willinger and D. V. Wilson, On the self-similar nature of Ethernet traffic, in D. P. Sidhu, ed., ACM SIGCOMM, San Francisco, California, 1993, pp. 183-193.

[18] T.-T. Y. Lin and D. P. Siewiorek, Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis, IEEE Transactions On Reliability, 39 (1990), pp. 419-432.

[19] S. Ma and J. L. Hellerstein, Mining partially periodic event patterns with unkonow periods, 17th International Conference on Data Engineering, Heidelberg, Germany, 2001, pp. 205-214.

[20] M. Malhotra and A. Reibman, Selecting and implementing phase approximations for semi-markov models, Stochastic Models, 9 (1993), pp. 473-506.

[21] J. K. Muppala, G. Ciardo and K. S. Trivedi, Stochastic Reward Nets for Reliability Prediction, Communications in Reliability, Maintabinabiliity and Serviceability, 1 (1994), pp. 9-20.

[22] J. C. Principe, A. Rathie and J. M. Kuo, Prediction of chaotic time series with neural networks and the issue of dynamic modeling, Int. J. of Bifurcation and Chaos, 2 (1992), pp. 989-996.

[23] X. Ren, S. Lee, R. Eigenmann and S. Bagchi, Resource Failure Prediction in Fine-Grained Cycle Sharing System, IEEE HPDC, Paris,France, 2006.
[24] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta and A. Sivasubramaniam, Critical event prediction for proactive management in large-scale computer clusters, KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press, Washington, D.C., 2003, pp. 426-435.
[25] K. S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications, John Wiley and Sons, 2001.
[26] W. Willinger, V. Paxson and M. S. Taqqu, Self-similarity and heavy tails: structural modeling of network traffic, (1998), pp. 27-53.
[27] R. Wolski, N. T. Spring and J. Hayes, Predicting the CPU availability of time-shared Unix systems on the computational grid, Cluster Computing, 3 (2000), pp. 293-301.