

Improving Cluster Availability Using Workstation Validation

Taliver Heath, Richard P. Martin, Thu D. Nguyen
Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854
{taliver, rmartin, tdnguyen}@cs.rutgers.edu

Appears in Proceedings of the ACM SIGMETRICS Conference, June 2002.

ABSTRACT

We demonstrate a framework for improving the availability of cluster based Internet services. Our approach models Internet services as a collection of interconnected components, each possessing well defined interfaces and failure semantics. Such a decomposition allows designers to engineer high availability based on an understanding of the interconnections and isolated fault behavior of each component, as opposed to ad-hoc methods. In this work, we focus on using the entire commodity workstation as a component because it possesses natural, fault-isolated interfaces. We define a failure event as a reboot because not only is a workstation unavailable during a reboot, but also because reboots are symptomatic of a larger class of failures, such as configuration and operator errors. Our observations of 3 distinct clusters show that the time between reboots is best modeled by a Weibull distribution with shape parameters of less than 1, implying that a workstation becomes more reliable the longer it has been operating. Leveraging this observed property, we design an allocation strategy which withholds recently rebooted workstations from active service, validating their stability before allowing them to return to service. We show via simulation that this policy leads to a 70-30 rule-of-thumb: For a constant utilization, approximately 70% of the workstation failures can be masked from end clients with 30% extra capacity added to the cluster, provided reboots are not strongly correlated. We also found our technique is most sensitive to the burstiness of reboots as opposed to absolute lengths of workstation uptimes.

1. INTRODUCTION

Recent years have seen the proliferation of large-scale Internet services. In spite of current economic conditions, these services will continue to make inroads into everyday life. However, the construction of highly available Internet services is an inexact science at best. Complete and partial outages due to failures are still common. Indeed, failures of these services are now important enough to make front-page news [8, 13, 20].

For years, however, highly available systems have been produced in a variety of contexts such as aerospace and manufacturing. Com-

mon to these domains is a general 3 step framework [24]: (1) decompose the system into components isolated by well-defined interfaces; (2) characterize the fault behavior of each component as an isolated unit; and, (3) architect the system to tolerate expected failure scenarios based on components' failure behaviors. The design of RAID systems is one example where a similar approach has been used successfully in the construction of highly available storage systems.

In this work, we apply the above framework to the construction of highly available Internet services. In particular, we identify the commodity workstation as a critical component, focus on characterizing its failure behavior, and present and evaluate a method for hiding node failures from clients of an Internet service. We are motivated to focus on commodity workstations because many Internet services run on workstation clusters [2, 10]¹. Further, a workstation is often the *unit of replacement* as failures occur. That is, when a sub-component such as a disk or card of a workstation fails, the entire workstation must be shut down and taken out of the cluster. When the operating system of a node crashes, all resources of the node are lost to the service while the node is rebooting. Thus, we view a workstation as a single black-box component of these services.

Unfortunately, the definition of what constitutes a failure of a workstation can be ambiguous. For example, is a failed disk-request, retried by the operating system, a failure? What about an OS memory leak that makes socket connections painfully slow? We circumvent this ambiguity by defining failure as the shutdown or crash of a workstation node, regardless of the cause. Thus, the central metric that we concentrate on is the Time-To-reBoot (TTB): that is the time between pairs of adjacent reboots of a machine². TTB is an appropriate metric in spite of its broad scope—it aggregates the failure and repair rates of myriad sub-components such as the operating system, the hardware, and the operator, into a single metric—because, as already discussed, a node is often the unit of replacement for cluster-based services.

Having defined the component of interest, what constitutes failures,

¹Of course, cluster-based systems include other important components such as network-attached disks, hubs, switches, and cables. Characterizing these components is beyond the scope of the immediate paper but is clearly important to the overall application of the framework to the design of Internet services.

²In actuality, a better metric is Time-To-Failure (TTF): that is, the time from when a machine starts until it reaches some definition of failure (e.g., a clean shutdown or crash). Unfortunately, the system logs did not contain sufficient information to compute TTFs. Thus, we were forced to approximate TTF with TTB.

and a metric, the next step in our methodology is to characterize the component's failure behavior with respect to the metric. A key advantage of our definitions is that we reduce this characterization to a straightforward quantification of the statistical distribution of TTBs. We analyze the `last` logs of 3 clusters operating in distinctively different environments to gather empirical TTB data. These three clusters include 18 remote access workstations supporting a general academic research workload, 19 general access workstations supporting undergraduate laboratory projects, and 89 workstations supporting a medium-scale Internet service. The obtained logs vary in length from slightly over one month to over three years, giving a total of 3343 observed reboots.

Next, we fit the observed distribution of TTBs for each cluster to a number of theoretical distributions and compute which provides the best fit. Our analysis shows that the TTB for all 3 clusters is best modeled as a Weibull distribution with a shape parameter between 0.33 and 0.49. This has important implications. First, the distribution is not memory-less, suggesting models using exponentially distributed times may not adequately describe node failure behaviors. Second, a shape parameter of less than 1 implies that *workstations become more reliable with time*. That is, we can model a workstation as a component that becomes increasingly reliable the longer it has been operating.

Finally, we propose one possible resource allocation policy that leverages the observed failure characteristics of cluster nodes to hide node failures from end clients. In particular, given a cluster used to support some Internet service, we divide the available machines into two pools: the *stable* pool and the *proving ground* pool. Each incoming client request is directed to a machine in the stable pool by a front-end load-director. When a machine in the stable pool fails, it is moved to the proving ground. As the size of the stable pool decreases (because of reboots and subsequent migration of nodes to the proving ground), the overall load will increase. When the load crosses a threshold, we choose the workstation node that has been up the longest in the proving ground and move it back into the stable pool.

Our allocation policy elucidates the fundamental trade off between excess capacity and improved availability. As excess capacity increases, rebooted machines can be kept in the proving ground for longer periods of time. According to our observation of increasing reliability of nodes with uptime, the longer we can keep a machine in the proving ground, the less likely it will fail when brought into service in the stable pool. Thus, many failures should occur while machines are in the proving ground, where failures are masked from clients because these machines are not being used to process real client requests. Of course, the machines in the proving ground should be placed under some test load, possibly a duplicate of some part of the real load, as utilization is likely a significant factor for node failures.

We evaluate the above allocation policy by simulating medium-sized clusters (100 nodes), using our fitted distributions to generate reboot events. To validate these results, we also simulated smaller clusters, using our actual trace data from the observed clusters to generate reboot events. We found that results derived from these two sets of experiments were mostly consistent, lending confidence to the modeling of workstation uptimes as a Weibull distribution.

Our results lead to a 70-30 rule-of-thumb: if 30% excess capacity is available, it can be used to mask approximately 70% of the

node failures from end clients provided reboots are not highly correlated. As excess capacity increases to 50%, as much as 85% of the node failures can be hidden from end clients. As shall be seen, however, correlation in reboots reduce the effectiveness of our validation technique. Since correlated failures in general make it more difficult to achieve high availability—for example, no fault-tolerance technique except site-replication will tolerate the failure of the entire cluster due to external events such as a power or cooling outage—service designers and operators should strive to avoid correlated failures. Such efforts would maximize the benefits of our validation-based technique.

Our analysis has two potential sources of inaccuracies: (1) TTB includes the Time-To-Repair (TTR), which may distort our characterization of nodes' uptimes, and (2) it does not differentiate between when a node is up and functioning correctly as opposed to when a node is up but its functionality is impaired because it is in a non-crashed fault state. For example, slow socket connections or degraded file system performance because of intermittent disk failures. The fact that the failure distributions from all three observed clusters are best described by the Weibull distribution with a shape < 1 gives us some confidence that (1) is not a source of significant distortion; TTR would have had to affect data from each cluster in the same way even though the clusters were operating in very different environments. The only real way to address this concern, however, is to obtain better data. In Section 7, we propose some ideas on how to record more information. We believe that the second issue is not as serious. All three clusters that we studied were under 24x7 maintenance. Thus, machines were unlikely to be in severe non-crashed fault states for very long before they were rebooted.

Despite the imprecisions introduced by the quality of data we were able to obtain, we make several key contributions. First, we argue that the entire workstation node is a primary component whose failure behavior we should understand, rather than that of the pieces such as controllers, network interfaces, and operating systems. We show that a gross measure such as TTB of workstation nodes is good enough to improve service availability. In fact, we believe that such a gross measure is actually more useful than a set of finer-grained measures because it is more difficult to incorporate the latter into a design. Second, we show that data from clusters operating in very different environments all suggest a distribution of uptime that is not exponential and thus is not memory-less. Finally, we show that appropriate policies can be designed and implemented to take advantage of an understanding of isolated component failure characteristics, provided the components have fault-isolated interfaces.

The remainder of the paper is organized as follows. Section 2 describes the three clusters and our reboot measurement methodology. In Section 3, we match the observed distributions to several theoretical distributions and show how a Weibull is the best fit. Section 4 describes our resource allocation algorithm and evaluates it using simulation. In Section 5, we discuss how the Weibull shape impacts strategies to improve availability. Section 6 discusses related work and Section 7 concludes with future work and implications.

2. MEASURING THE TIME BETWEEN REBOOTS

As part of the first step in our methodology, we must measure the actual TTB of workstation clusters. We analyzed the `last` logs

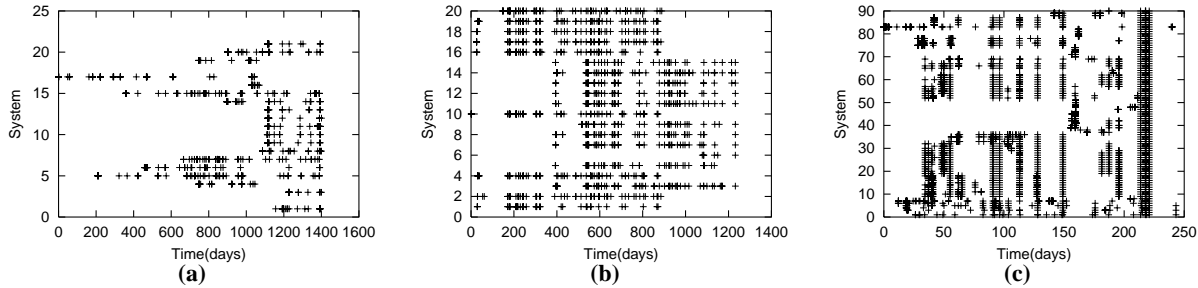


Figure 1: Reboots as a Function of Time. This figure shows the raw reboots for (a) Machine Room Cluster, (b) Undergrad Cluster, and (c) Internet Service Cluster. Time is represented on the X-axis, and machine number on the Y-axis. A point is plotted each time a reboot occurs. The leftmost point of each line is the first data point that was available for that machine. Time lines for different machines in a cluster are synchronized so that vertical lines imply correlated failures across machines.

Cluster	Shortest Log	Longest Log	Average Log	Reboots Observed
Machine Room	39 days	1178 days	481 days	521
Undergraduate	153 days	871 days	794 days	979
Internet Service	15 days	240 days	141 days	1843

Table 1: Reboot Log Coverage. This table shows the minimum, maximum and average coverage periods of the reboot logs for the 3 clusters. The table also shows the average total number of reboots observed.

from three clusters operating in different environments:

Machine Room Cluster This cluster is comprised of 18 machines that were physically housed in our machine room and were accessible only via remote login to faculty and graduate students. 16 of these machines were Sun Ultra-1 workstations and 3 were Sun Sparc-20 workstations. These machines were managed by several different administrators who did not necessarily coordinate maintenance schedules. The machines ran a mixture of Solaris 2.7 and 2.8.

Undergraduate Cluster This cluster is comprised of 19 Sun Ultra-1 workstations running Solaris 5.8. All machines were physically housed in one laboratory and used exclusively by junior and senior undergraduate computer science students for projects in upper division courses. A single system administrator manages the entire cluster.

Internet Service Cluster This cluster is comprised of 89 machines running Solaris 5.8. 43 of the machines were Ultra 5 workstations, 4 were quad-processor Sun Enterprise 450's and the remaining 32 were Netra "T1" thin-cluster boxes. All were physically housed in the machine room of an Internet service company which had considerable physical security. A team of 3 system administrators was responsible for managing this cluster.

We intentionally collected data from clusters operating in environments with very different characteristics, such as accessibility (machine room or open), the number of administrators, the type of institution (university and corporate) and finally, the target user group (faculty, undergraduates, and the general public). We show that in spite of the many differences, all three clusters exhibit similar reboot behavior. We are thus confident our results are widely applicable.

Whenever a Unix machine boots, it records the time in the last log. We compute a set of TTBs from each sequence of boot times, $T_{b0}, T_{b1}, T_{b2}, \dots$, given by a last log as $TTB_i = T_{b_{i+1}} - T_{b_i}$. As already discussed, there is a discrepancy between our computed TTBs and actual node uptimes, as the TTBs include repair times. We were unable to remove this inaccuracy because most machines did not record the crash/halt/shutdown time when they went down.

The logs vary in the amount of time covered, ranging from about 1 month to over 3 years; all time periods were in 1997 to 2000. Figure 1 plots the raw reboot data; Table 1 shows the length of time for the logs and the number of reboots that we recorded for each cluster.

3. THE SHAPE OF FAILURE

After gathering the raw TTB data, our next step is to see whether and how well the measured data fits some theoretical distribution. Figure 2 plots the cumulative fraction of TTBs less than or equal to a time period t vs. time. These curves are, in effect, the Cumulative Distribution Functions (CDF) of our measured data, assuming reboots are completely independent. Our task is thus to fit the curves in Figure 2 to known CDF's. (Figure 2 also shows the CDFs of the fitted Weibull distributions superimposed on the observed data, since, as shall be seen, the Weibull best describes our empirical data.)

We fit the observed data for each cluster to the CDFs of several theoretical distributions, including exponential, Weibull, Pareto, and Rayleigh. For each theoretical distribution, we used the Maximum Likelihood Estimation (MLE) [19] to compute the needed parameters (to match the distribution to the empirical data). We then computed quantile-quantile plots [15] for each pair of empirical data and theoretical distribution. If the theoretical distribution matches the empirical data, the quantile-quantile plot would give a straight line (intercept 0, slope 1). Any variation away from this straight

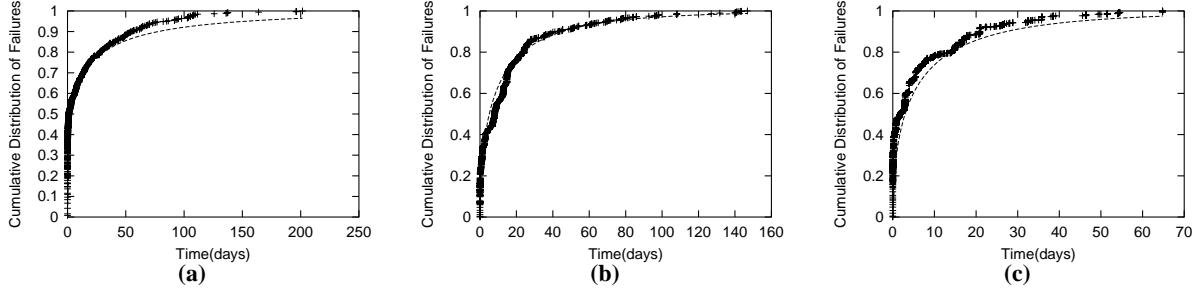


Figure 2: TTB distributions. Cumulative percentage of TTBs less than or equal to time t vs. time for (a) Machine Room Cluster, (b) Undergrad Cluster, and (c) Internet Service Cluster. For each cluster, the CDF of the MLE fitted theoretical Weibull distribution is also shown superimposed on the observed data. The data points are plotted as + symbols, while the fitted curve is shown as a dashed line.

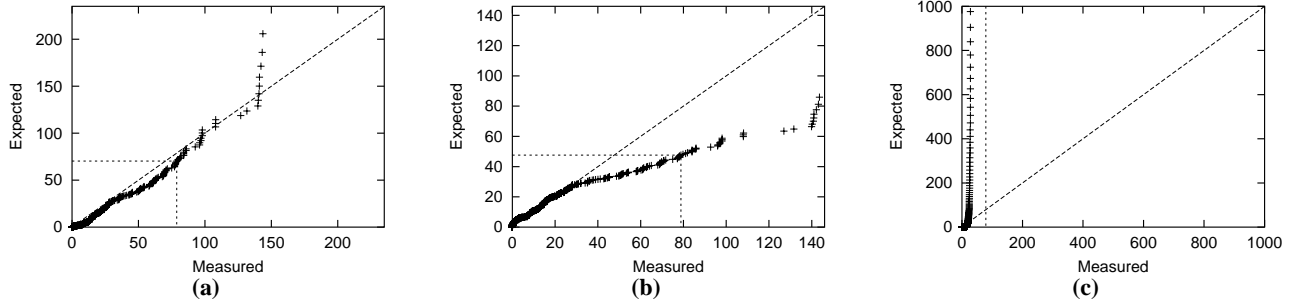


Figure 3: Fitting the Weibull distribution to the observed TTBs. Quantile–quantile plots for the Undergrad cluster using the MLE parameters for (a) Weibull, (b) Exponential, and (c) Pareto distributions. Unit for both axes is day. In all three graphs, the diagonal dashed line represents the $y = x$ line which the quantile–quantile data should approximate. The dotted line in the graphs shows the 95th percentile for the data.

	Shape	Scale (minutes)
Machine Room Cluster	0.33	6000
Undergrad Cluster	0.49	13200
Internet Service Cluster	0.41	3850

Table 2: Fitted Shape and Scale Parameters. This table shows the resulting shape and scale parameters of the Weibull distributions fitted to the observed data.

line indicates a difference between the observed data and the theoretical distribution. Finally, we decided which theoretical distribution best describes our empirical data, and whether it provides a reasonable model, by visual inspection of the quantile–quantile plots. (For choosing between the Weibull and exponential, the MLE computation serves as a validation of our visual inspection. A Weibull distribution with shape parameter of 1 is an exponential distribution.)

The above process led us to conclude that the Weibull distribution best describes the reboot behaviors of all three clusters. Figure 3 gives the quantile–quantile plots of the empirical distribution vs. the Weibull, exponential, and Pareto theoretical distributions for the Undergrad Cluster. Quantile–quantile plots for the other two clusters are very similar. Clearly, Pareto is not a good fit (Rayleigh is equally inaccurate and so is not shown in the interest of saving

space). Comparing Weibull against exponential, we observe that the Weibull fits the empirical data much more closely until the 95th quantile, after which, the theoretical distribution starts to give much longer TTBs than we observed. Table 2 gives the MLE parameters of the matching Weibull for all clusters, providing further evidence that the Weibull is a better fit for our data than exponential; all shape parameters are well below 1. Finally, Figure 2 shows the good fit of the Weibull distribution by superimposing the theoretical CDFs on the empirical ones.

Implications of the Distribution. An important question in the design of highly available systems is *if a particular component has been operating for time t , what is the likelihood that it will fail at time t ?* We can answer this conditional probability by examining the *hazard rate* of our known distribution. The hazard rate for any distribution is:

$$h(t) = \frac{pdf(t)}{1 - cdf(t)}$$

where $h(t)$ represents the conditional probability density that a t -unit-old item will fail vs. time [19].

The *pdf* and *cdf* of the Weibull distribution are as follows:

$$pdf(t) = \frac{shape}{t} \left(\frac{t}{scale}\right)^{shape} e^{-\left(\frac{t}{scale}\right)^{shape}}$$

$$cdf(t) = 1 - e^{-\left(\frac{t}{scale}\right)^{shape}}$$

leading to a hazard rate of:

$$\frac{shape}{scale} \left(\frac{t}{scale}\right)^{shape-1}$$

This hazard rate has the following implications. If the Weibull *shape* is 1, then the hazard rate is constant and so the chance of rebooting is independent of how long the node has been up. If *shape* > 1, then a reboot becomes increasingly likely as time passes. On the other hand, if *shape* < 1, which is true for all of our clusters, *reboots become less likely the longer a node has been up*.

This increasing reliability may seem counter-intuitive. However, many plausible scenarios can explain the observed hazard rates such as: (a) when a machine is taken down for a hardware upgrade the administrator may introduce configuration errors, requiring several rounds of reboots before the machine becomes stable again, (b) new hardware components often have a “burn-in” time during which the probability of failure may be significantly higher than after they have been operating for a while, (c) installation of new software or patches may introduce bugs, leading to frequent crashes until the administrator finds a stable configuration, and (d) operator confusion about configuration changes (such as mount-points) may lead to several rounds of reboots until the correct setting is found.

Assumption of Independence. Our distribution-fitting methodology assumes reboots are independent events. However, visual inspection of Figure 1 would lead an observer to think that some dependency likely exist; for example, the vertical lines suggest reboots are often correlated across multiple machines. In the Internet Service cluster, for example, correlated reboots occurred often due to the failure of the air-conditioning system. Still, this does not preclude the theoretical distribution from being useful. What it does imply is that we must carefully validate that policies formulated from properties of the theoretical distribution work well in practice. In Section 4, where we propose and evaluate one such policy, we validate its practicality by using trace-driven simulation in addition to relying on synthetic reboot sequences generated from the theoretical distribution.

4. IMPROVING CLUSTER AVAILABILITY

Having characterized the failure characteristics of commodity workstations, the next step in our methodology is to leverage this knowledge to increase the availability of cluster-based services. The fact that TTB for nodes in all three observed clusters are best described by Weibull distributions with shape parameters less than 1, and the corresponding monotonically decreasing hazard rate, leads to the following heuristic:

After a reboot, a machine should not be placed into active service until it has proven to be stable by operating for some time without requiring another reboot.

Of course, a machine operating in such a *validation* mode must be loaded somehow; likely, some duplication of the real load would

make the best validation load because that is what the workstation would be supporting when put back into service. Once a machine has proven to be stable by being up for a while, it can be placed back into active service with confidence that, likely, it will not need to be rebooted soon.

This heuristic is quite intuitive and is already being used by system administrators everywhere. However, previously, it was impossible to answer the question: *how long should a machine remain in validation mode before it is placed back into active service?* In this section, we present a resource allocation policy that provides an automatic and dynamic answer to this question. We then follow with an evaluation of the policy’s effectiveness using simulation.

4.1 A Validation-Based Policy

Assuming that cluster nodes are identical in that any node can service any client request, we propose the following resource allocation policy to mask node failures from end clients:

- Divide cluster nodes into two logical pools, the *stable* pool and the *proving ground* pool. The original division can be arbitrary except that the stable pool must start with at least one machine.
- Direct client requests only to machines in the stable pool.
- If the load on the stable pool exceeds a predefined threshold, choose the node that has been up for the longest time from the proving ground pool and move it to the stable pool. Repeat until the load drops below the threshold. The threshold can be specified using a variety of metrics depending on the actual configuration of the cluster. Possible metrics include average number of active client requests per stable node, queue length at the front-end load-director, throughput of the stable pool, or average latency for servicing client requests.
- When a node in the stable pool fails, remove it from the stable pool, repair and reboot. When the node is back in operational mode, move it to the proving ground pool.
- When a node in the proving ground pool fails, repair and reboot. When the node is back in operational mode, leave it in the proving ground pool, resetting its uptime to 0.

Recall that our definition of failure includes the orderly shutdown (and subsequent reboot) of a machine as well as operational crashes. This means that even shutdowns of machines for patches, upgrade, etc. are covered by our validation policy. One may ask why this is so. Wouldn’t the system administrator hold the machine out of operation until it could be validated somehow already? Yes. However, currently, this validation is completely ad hoc. The load-driven migration of machines between the proving ground and the stable pool gives system administrators a well-defined validation process. In practice, we expect system administrators to perform whatever validation they are doing now after modifying the configuration of a machine. Once done, they would put this machine into the proving ground pool for final validation³.

³Current ad-hoc validation methods are often insufficient. For example, numerous times a system administrator has told one of the authors that a machine is operational when it is still mis-configured. We strongly believe that validation under realistic load in the proving ground pool is necessary regardless of the cause for taking a machine out of active service.

4.2 Evaluation Metric

Before evaluating our validation-based policy, we first consider the question of which metric to use. We originally used *yield*, first proposed in [9], which is the percentage of successful requests. Yield, however, turned out to be too coarse of a metric. In particular, yield is very sensitive to the time interval over which it is measured. For example, a service which loses 100 requests evenly spread over 100 hours is much different than one that loses 100 requests in one minute and none for the remaining 99 hours and 59 minutes. Despite this difference, both have equal yields over the 100 hour interval.

The second problem with yield is that one must make many assumptions about the cluster design and data-partitioning in order to compute it. While an Internet service designer must take all of these into account, the focus of our work is on masking node failures from end clients. Thus, we decided that the percentage of reboots not observable by end clients is a more appropriate metric and is the measure used in the remainder of this section.

The percentage of reboots masked from end clients is easily computed as the number of reboots that occur in the proving ground pool divided by the total number of reboots. As we scale load, we can generate an *avoidance curve*. Armed with an avoidance curve and enough data about actual numbers of reboots, a designer can compute other metrics such as harvest [9], yield and availability given a data-partitioning scheme and the parametric assumptions in Table 3.

4.3 Simulation Model

We use a simple abstract model of cluster-based Internet services, much like the one in [2]. The model is comprised of a front-end load-director and a cluster of back-end nodes. The front-end distributes client requests to the back-end nodes, using the queue lengths at the back-end nodes as the means for load balancing. The front-end never fails. A back-end node can only service one request at a time; this is a simplification from real practice but one can think of each request as representing a batch of client requests that is serviced concurrently. Because our metric only centers around where reboots occur, this simplification does not affect the validity of our simulation. Again, we assume that data (and other resources) are structured in such a way that any back-end node can service any client request.

Our event-driven simulator uses a file of reboot events to describe the *upset load*, or sequence of fault-events. By using input files, either real traces or synthetic reboot events can be used in the same simulation environment. Real traces come from processed logs, while synthetic event sequences are generated by a small external program.

Arrival of client requests is exponentially distributed with the average rate set to achieve a particular average work load on the service. Request service times are also exponentially distributed. Node repair time is a constant. The lag time for the load-director to register a node failure is also constant. Table 3 summarizes these parameters.

To keep the simulation from becoming overly complex, we use the known average load to drive our validation-based allocation policy. In particular, we use the following algorithm:

Parameter	Distribution	Value
Load	Poisson	Varies (% total capacity)
Service	Poisson	120 per minute
Repair	Constant	2 minutes
Lag	Constant	10 seconds
Failures	Weibull	Varies (according to shape and scale)

Table 3: Simulation Parameters. This table shows the constants and distributions used in our event-driven Internet service simulation.

When a Failure Occurs

If the Failed Machine is in the Stable Pool **then**

Move the Machine to the Proving Ground Pool

Begin Repair and Reboot of the Machine

If $Size_{ProvingGroundPool} > Pool_{Max}$ **then**

Move the Machine with the greatest uptime from the Proving Ground Pool to the Stable Pool

where the maximum size of the proving ground pool is:

$$Pool_{Max} = (1 - (\bar{x}_{load} + 3\sigma_{load})) \cdot NumberOfNodes$$

\bar{x}_{load} is the average load and σ_{load} is the standard deviation of the load. We use the average load plus 3 standard deviations to ensure that the size of the stable pool is sufficient to service bursts of requests.

4.4 Evaluation

We performed four sets of experiments to study the effectiveness of our technique. We investigated the impact of changing the shape, changing the scale, changing the cluster size, and finally, compared the effectiveness of our technique on a real trace file to the synthetic upset load.

In the first set of experiments, we simulated the operation of clusters of 100 nodes for 200,000 minutes (approximately 139 days). We generated a random upset load from a Weibull distribution with a constant scale of 6000 minutes and varied the shape parameter. Table 4 gives the shape parameters that we studied and the corresponding numbers of reboots.

In the second set, we generated random reboot traces from a Weibull distribution with a variable scale parameter and a constant shape of 0.41; 0.41 is the average of the shape parameters for our empirical data (see Table 3). We again simulated clusters of 100 nodes for 200,000 minutes. Table 4 gives the scale parameters that we studied and the corresponding numbers of reboots.

In the third set, we studied the effect of cluster size while keeping a constant shape (0.41) and constant scale (6000) for the upset load's distribution. We simulated clusters of different sizes under different constant loads for 200,000 minutes.

In the final set of experiments, we used the real reboot traces from the three observed clusters to assess the differences in the performance of our policy arising from the differences between the real reboot data and a synthetic upset load generated from known Weibull distributions. Although a trace-driven simulation is more realistic than using a theoretic distribution, it has the disadvantage

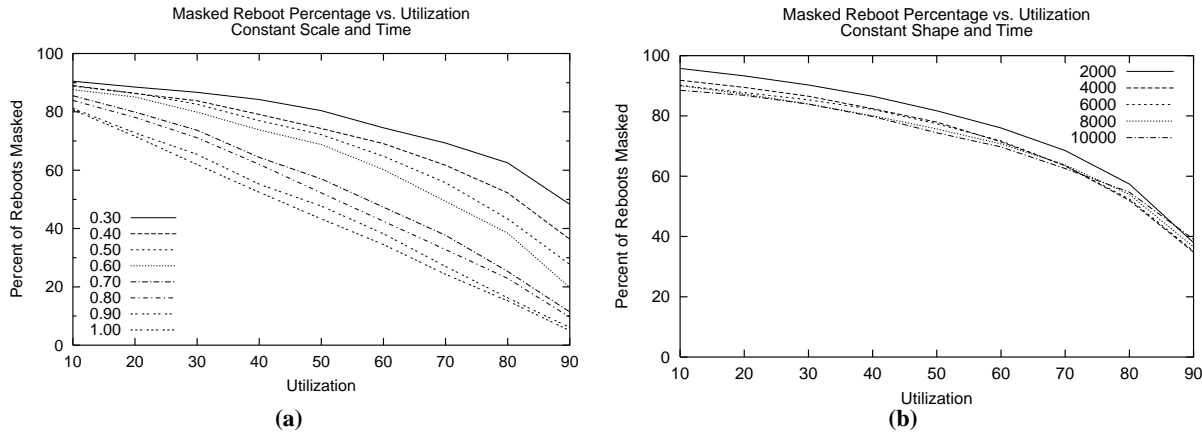


Figure 4: Percentage of failures masked successfully. Figure (a) shows percentage of failures masked successfully vs. load as the shape parameter is varied for a constant scale of 6000 minutes; Figure (b) shows the percentage of reboots masked successfully vs. load as the scale is varied for a constant shape of 0.41. The Figure shows that our technique is sensitive to the shape and insensitive to scale, implying that absolute uptime is not as critical as predictability.

Scale = 10000		Shape = 0.41	
Shape	Reboots	Scale	Reboots
0.2	409	2000	3740
0.3	673	4000	1750
0.4	843	6000	1340
0.5	1062	8000	1062
0.6	1382	10000	852
0.7	1719		
0.8	1873		
0.9	1876		
1.0	1897		

Table 4: Number of Reboots. This table shows how the number of reboots changes for a constant 200,000 minute interval as we (1) keep the scale constant at 6000 minutes and change the shape and (2) keep the shape constant at 0.41 and vary the scale.

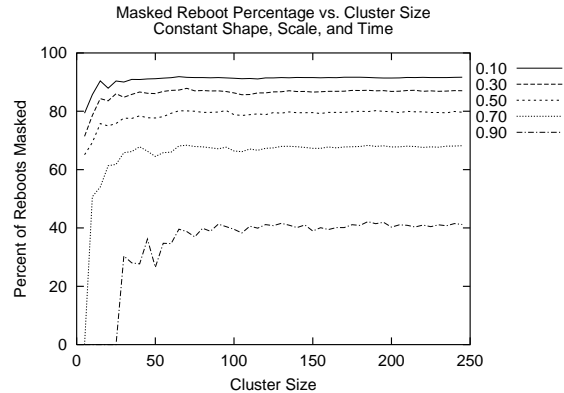


Figure 5: Effects of Cluster Size. This figure shows the percentage of reboots masked successfully vs. cluster size at different utilizations for using a synthetic upset load.

that we can not scale our experiments beyond the observed cluster size and time period. Also, studying the effects of scale and shape would be difficult given the limited data. In addition, real traces contain external correlation effects that a designer may be able to mask. We use results derived from trace data as validation of results from the first three sets of experiments.

Synthetic Upset Load. Figure 4 shows the results of the first two sets of experiments. Figure 5 shows the results of the third set of experiment. Based on these results, we make the following observations.

Validation-based allocation policies can successfully mask significant numbers of reboots. For example, for a shape of 0.4, which is representative of our observed clusters, our policy masks close to 75% of the reboots at 70% load. When the load increases to 80%, our policy still masks over 60% of the reboots.

Designers can take away a general 70-30 rule-of-thumb. That is, a service that has 30% extra capacity in reserve can mask approximately 70% of the reboots from end clients, provided reboots are

not correlated. The “70” will change depending on the exact characteristics of a particular cluster but is roughly accurate based on the scale parameters we observed.

Availability is more sensitive to how well failures are predicted as opposed to the absolute length of the uptimes. Figure 4 shows that our policy is much more sensitive to the shape parameter than the scale parameter. This result shows that improved overall system availability can come from concentrating on predicting failures rather than simply improving component lifetimes. Recall that the smaller the shape parameter, the higher the burstiness of the reboots. Thus, the less the failure process exhibits memory-less behavior, as characterize by the shape, the better our algorithm can predict the likelihood of the next reboot, and the more effective the proving ground will be.

There must be sufficient excess capacity for validation-based policies to work. There is a clear regime of operation where the avoidance curve changes slowly with offered load. For the shape pa-

Cluster	Size	Simulated Time (days)	Total Reboots	Shape	Scale (days)
Machine Room	13	244	123	0.39	13175
Undergraduate	18	715	648	0.55	14850
Internet Service	56	181	904	0.54	8375

Table 5: Trace-Driven Simulation Parameters. This table shows the cluster sizes, time periods, and total reboots events we were able to simulate using a trace-driven methodology. We dropped several machines from each cluster because their logs were much shorter than the rest, and so would have shortened the simulated time too much.

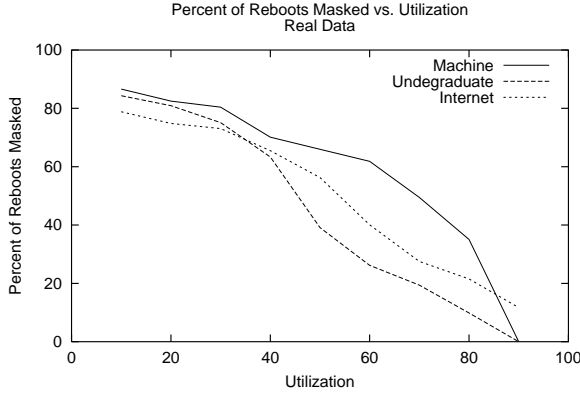


Figure 6: Failures Masked for Trace-Driven Simulation Experiments. This figure shows the percentage of reboots masked successfully vs. load when failures are driven by our collected traces. Note that in this case we ignore the first, and therefore non-maskable, reboot of each machine in the cluster. This allows us to more closely approximate a steady-state system with our limited trace data.

rameters of interest, the stable operating regime ends around 80% utilization. Beyond this inflection point, the avoidance curve drops sharply. This is as expected because the stable pool must have some excess capacity to ensure reasonable service in the face of bursty arrival of client requests. If there is little excess capacity, machines will quickly migrate to the stable pool after a reboot, limiting the effects of our policy.

Our technique is most effective when the cluster contains at least 15 machines. This effect can be seen in Figure 5, where there is a sharp drop at the left, even for modest utilizations. Cluster size matters because workstations are moved between the two pools in discrete units of entire workstations. The larger the cluster size, the larger the proving ground pool. The larger the proving ground pool, the more choices we have between validated workstations to move back to the stable pool when necessary.

Trace-Driven Upset Load. An advantage of our simulator design is that we can use actual reboot traces to drive a simulation, allowing us to verify that our policy would be effective in practice. In order to capture real correlation effects, which will degrade the performance of the proving ground, we used a fix window of real-time to generate an upset load instead of normalizing the time of each log to the same hypothetical starting point. Because the logs

varied in length, we thus used only a portion of each cluster trace.

In choosing the time window we had to balance between maximizing the cluster size and maximizing the time period (to observe steady state behaviors). Table 5 gives the parameters for our trace-driven simulations. Figure 7 gives an close-up view of a portion (100 days) of the raw data that we selected for two clusters. Table 5 shows the shape parameters for the Undergraduate and Internet Service clusters are substantially higher (0.55 vs 0.41) for the chosen time period.

Figure 6 shows the avoidance vs. utilization curves for our three clusters. Observe that all three avoidance curves follow the expected trend. For each curve, there is a stable operating regime followed by a sharp drop where the load exceeds the algorithm’s ability to mask failures. However, the performance of our technique is worse than predicted by our earlier simulations in all cases.

Two effects can account for this discrepancy between the synthetic and trace-driven upset loads. First, the small size of some of the clusters will degrade the performance. Even for the synthetic upset loads, Figure 5 shows a sharp drop in the avoidance curve below 15 workstations.

A second, more important effect is the degradation due to correlation. If many workstations fail at once, there may not be enough excess capacity in the proving ground compared to when reboots are not correlated. For example, an omitted component or factor from our model, such as an air conditioning unit, may cause many reboots at once. Such omitted components may make our policy ineffective at a given load because a single reboot signifies the loss of a large portion of the nodes closely spaced in time. Many machines will have to be rushed back into the stable pool before they have shown themselves to be stable.

Although the Machine Cluster had the smallest size (13), careful visual inspection of Figure 7(b) reveals that this cluster has the least correlation. When coupled with the fact that this cluster has the smallest shape parameter, this is consistent with the Machine Cluster giving the best performance and the smallest rate of performance degradation as load increases. In this case, our rule-of-thumb changes from 30-70 to approximately 40-60; that is, 40% excess capacity can be used to mask upwards of 60% reboots from end clients.

The avoidance curve for the Internet Cluster starts out much as expected, slightly worse than predicted when using the theoretical distribution. Around 50% utilization, however, performance degrades much more rapidly than predicted. We suspect that this is due to the very high correlation between reboots in this cluster; even casual inspection of Figure 1 shows many vertical bands, suggesting a high degree of correlation. The authors know that, in this case, much of the correlation in reboots was caused by air conditioning failures in the machine room. Whenever one of a pair of air conditioning units failed, the system administrator powered off most of the cluster. This correlation makes it difficult for any fault-tolerance technique which focus only on the workstations and their sub-components, including our approach, to mask failures. A better method in that case would expand the set of modeled components to include the air conditioners.

The avoidance curve for the Undergrad Cluster shows the same behavior as the Internet cluster although the inflection point comes

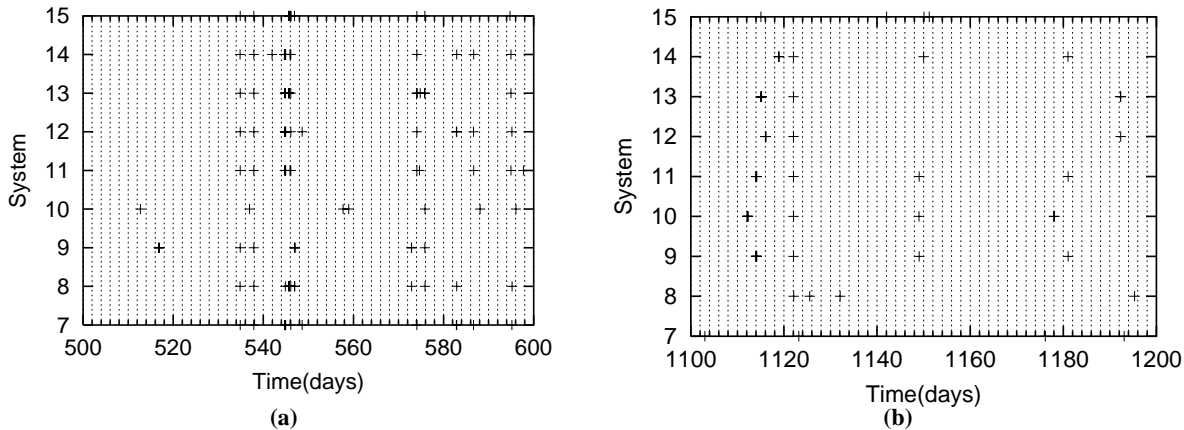


Figure 7: Enlargement of Timeline. Figure (a) shows an enlargement of 100 days of the Undergraduate Cluster, while Figure (b) shows a similar enlargement of the Machine Room Cluster. Notice that more events are aligned in the Undergraduate Cluster, indicated a higher degree of correlation.

even earlier and the performance degradation is sharper. We believe that this is due to high correlation between reboots. Figure 7(a) shows a number of vertical bands, again suggesting reboots were highly correlated. Some factors causing correlations for this cluster may have been that it was maintained by a single administrator in an environment where it was most conveniently to shutdown the entire cluster for events such as upgrading of the operating system. Also, the relatively smaller cluster size (18 machines compared to 56 for the Internet Cluster) further reduces the ability of our validation algorithm to mask faults.

Despite the differences, however, our results here are quite encouraging because they show that our policy can still mask a significant percentage of the failures despite the effects of smaller cluster sizes, and, more importantly, potential correlation in reboots, as long as the correlation is not overwhelming.

5. DISCUSSION

There are three broad approaches permeating the design space to mask component failures: (1) validation, (2) rejuvenation and (3) replication. In this section, we argue that the choice of approach is likely dependent on the nature of component behavior. In all cases, excess capacity is required to mask the failures; what is distinct are the underlying assumptions about component behavior and using that knowledge to predict those failures.

In validation based approaches, the components are prevented from acquiring system load until they have crossed a critical uptime threshold; they must pass a “burn-in” period. Time is used as a failure predictor, where increasing uptime leads to decreasing likelihood of failure. Our validation-based policy is an example of this technique applied to workstation clusters. Such techniques are most effective when the component’s hazard rate is decreasing with time, as is true with respect to workstations.

On the other hand, rejuvenation techniques are most appropriate when the hazard rate increases with time. If components behave in this manner the system should be designed to avoid component “burn-out” rather than “burn-in.” Several works have examined this technique viewing the Operating System as a component (see Section 6).

Replication based approaches are most appropriate when the failure distribution is exponential, and hence, memory-less. Under these conditions, time cannot be used to predict component behavior. Thus, unless there’s another parameter that can be used to predict component failures, replicating the work is the only way to bring about an increase in availability. Designs such as RAID take this approach for storage, using either total replication or error-correcting codes.

Of course, these techniques are not mutually exclusive. For example, it makes sense to use replication to hide failures that cannot be masked by our validation-based policy.

Finally, the choice of approach critically hinges on the definition of a component. In all complex systems, components are deeply nested. Choosing at what level to draw the line around component boundaries is currently more of an art than a science. In general, a component must at least have a well defined interface and failure semantics. In the context of Internet Service design, choosing the entire workstation node as the component and reboots as the failure semantic is a reasonable choice. However, future research is needed to guide the designer in choosing component boundaries because this implicitly permeates the resulting behavioral distribution, the ease and cost of validation, rejuvenation and duplication approaches for a given component. For example, in the Internet service context, replacing an entire rack-mounted workstation is easier, and much more cost-effective in operator time, than replacing an internal component such as the motherboard.

6. RELATED WORK

There is a vast amount of related work measuring software and hardware reliability and on schemes for improving system availability in many contexts. In this section, we focus on those contexts which are most applicable to clusters running scalable Internet services.

The closest work related to this study characterized the behavior of Microsoft Windows NT machines [25]. That study examined 2,127 reboots events of 503 machines recorded over a 4 month period. It is quite encouraging that the shape parameter of 0.49 for their fitted Weibull distributions for all failures is very close to our Weibull

distributions for Solaris clusters. In addition, they were able to use Windows NT event logging to delve deeper into the causes of reboots than we were able to with the Unix `last` logs. They found the majority of reboot events were caused by reasons other than hardware and software failures. They also observed correlated reboots. However, they did not extend the result to workstation clusters in an Internet Service setting. Also, even though they found Weibull shape parameters of less than 1, they did not use this information to propose any validation-based strategies for masking failures. Another study by the same group [16] produced a detailed state-machine model of a workstation node, and thus requires fairly detailed knowledge of Windows NT.

Another recent work examined the failure of components of an on-line image service [1, 21]. That work focused more on the failure rates of the node components rather than on examining the entire node. Interestingly, they found that commonly blamed culprits of failures, such as the operating system and the SCSI drives, were in fact quite reliable.

Much work has examined the causes of software failures. Typically, these look at characterize application bugs [6, 14] or operating system errors [7, 23]. These works differ fundamentally from ours in their choice of component. While the applications and operating systems are large components, from an Internet service perspective they are all nested in the larger component of the workstation node.

Rejuvenation approaches have been proposed in many different contexts. The original work [14] looked at telecommunication switches and long-running scientific programs. Another study examined rejuvenation in the context of clusters [22] and found time-based rejuvenation policies effective. However, their models assumed an increasing hazard function. Later empirical work by the same group found that the expected mean time for the operating system to run out of memory was 70 days [23], which is far longer than our observed average 14 day reboot interval. Other factors leading to rejuvenation being effective for operating systems require much greater mean uptimes. Finally, recent work has proposed extending the idea of rejuvenation throughout all layers of software [5]. While all these works will improve software robustness (and will likely mask software failures that do not lead to node reboot), our characterization shows that they are unlikely to improve the masking of node failures as the entire workstation node increases in availability with time.

The discrepancy between software failure and node reboots leads to a search for other causes. Studies of Tandem machines have suggested that in addition to hardware and software, the humans operators also play a key role in generating faults [11, 12]. However, the context of these works was limited to specialized machines in tightly controlled environments. Indeed, one paper [4] argues administrator and operator errors will soon become the dominant factor in component failures. More recent work in the Windows NT context validates these arguments [17, 18, 25]. Expanding on this theme, work has emerged using human-factor studies on to determine how actual humans behave in the context of RAID repair [3]. While our choice of workstation reboot metric capture human-factor effects to a large degree, our work leaves open the question if the human factor is dominant cause of the observed reboot distribution.

7. CONCLUSION

We have proposed the application of a general three-step framework to the construction of highly available Internet services. This framework is based on understanding the individual failure behavior of components isolated by well-defined interfaces. We have demonstrated the use of this framework by defining the workstation as one such component critical to Internet services, studying its failure behavior, and designing a resource allocation algorithm to improve overall system availability.

When viewed as a complete system of hardware and software, we have found no indication that a system that has been “up” for an extended period of time is more likely to be rebooted than any other system. In fact, our results show the opposite: for Sun workstations running various versions of Solaris in three distinct environments, a machine that has been up for a long period of time should be left up.

We demonstrated that a simple policy which takes advantage of understanding this behavior can mask a significant percentage of node failures from end clients. Our simulations show our validation technique results in a 70%-30% rule: by adding an additional 30% excess capacity, 70% of the workstation node reboots in a cluster can be masked from the clients provided the cluster is sufficient large (>15 machines) and reboots are not highly correlated.

Our trace-driving simulation results show that high correlation between reboots can significantly degrade the ability of our algorithm to mask faults. However, results for the cluster with the least correlation in reboots show that we should be able to achieve the above rule-of-thumb in practice. Reboots in this cluster were less correlated than in the other two because it was maintained by multiple administrators and did not suffer from external faults such as air conditioning failures. This points to the importance of ensuring non-correlated reboots; that is, the administrator of an operational cluster should not take the entire cluster, or even significant portions of the cluster, off-line for upgrades, patching, etc. Also, Internet service providers must take care to have adequate facilities so that external failures cannot lead to cluster failures.

Like rejuvenation techniques, our proving ground algorithm relies on predicting component failures using time as the predictor. However, our data shows that the assumed direction of the prediction behind rejuvenation techniques, that components decay over time, may be flawed, at least with respect to machines viewed in their entirety. Given our results and those in [12, 17, 25], it seems most probable that the countless number of event types that can lead to workstation “failures,” far outweigh the effect of individual component decay over time. However, our results did not cover time-scales on the order of many years. It may be the case that if we extended our observations to these time-scales, that we would see increasing failure rates. Given that the useful operating range of computer hardware is about 3-years, and software even less, such long regimes may be irrelevant to the construction of highly available services.

Another conclusion our results point to is that when a system starts, it may be far from a “clean” state. Indeed, configuration errors of different components may well cause the need for subsequent reboots, leading to the idea that a machine’s infant mortality period never fully goes away. This would also lend credibility as to why when a system has been up for a while, it tends to stay up. We have no evidence aside from our initial data of this claim being true, but

it does warrant further investigation.

Finally, future research in this area would be greatly aided by installing workstation nodes with much more detailed recording information; a "flight recorder" for the box. This observation was also noted in [25], where even with detailed knowledge of Windows NT event types 58% of all outages could still not be classified. We envision recording not just error messages, but also proactively recording machine load, configuration state, user and operator actions. For example, by using NVRAM and an OS heartbeat, we could not only record reboots but also accurately measure downtime. Using file fingerprinting techniques, we could record node configuration and application changes. Newer server machines have temperature and intrusion-detection sensors, and these should also be logged. Indeed, placing proximity detectors to record human movement near a machine may prove quite valuable in predicting when a machine might fail. Given the very low cost of storage today, it is possible to record many of these events for long periods. Analysis could then be performed both on a historical bases as well as make predictions about the stability of individual machines in real time.

8. REFERENCES

- [1] S. Asami. Reducing the cost of system administration of a disk storage system built from commodity components. Technical Report CSD-00-1100, University of California, Berkeley, 2000.
- [2] E. Brewer. Lessons from Giant-Scale Services. *IEEE Internet Computing*, 5(4):37–45, July/August 2001.
- [3] A. Brown and D. A. Patterson. Towards Availability Benchmarks: A Case Study of Software RAID Systems. In *2000 USENIX Annual Technical Conference*, June 2000.
- [4] A. Brown and D. A. Patterson. To Err is Human. In *First Workshop on Evaluating and Architecting System dependability (EASY '01)*, July 2001.
- [5] G. Candea and A. Fox. Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel. In *8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001.
- [6] S. Chandra and P. M. Chen. How Fail-Stop are Faulty Programs? In *1998 Symposium on Fault-Tolerant Computing (FTCS)*, June 1998.
- [7] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An Empirical Study of Operating Systems Errors, October 2001.
- [8] Claudia O'Keefe. Doomed by eBay. http://salon.com/tech/feature/2000/10/27/doomed_by_ebay/index.html, Oct. 2000.
- [9] A. Fox and E. Brewer. Harvest, Yield and Scalable Tolerant Systems. In *Proceedings of Hot Topics in Operating Systems (HotOS VII)*, Rio Rico, AZ, Mar. 1999.
- [10] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Scalable Cluster-Based Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Oct. 1997.
- [11] J. Gray. Why do Computers Stop and What Can Be Done About It? In *Proceedings Fifth Symposium on Reliability in Distributed Software and Database Systems*, Jan. 1986.
- [12] J. Gray. A Census of Tandem System Availability Between 1985 and 1990. *IEEE Transactions on Reliability*, 39(4):409–418, Oct. 1990.
- [13] J. Hu. Britannica.com crippled by user volume. <http://news.cnet.com/news/0-1006-200-920536.html>, Oct. 1999.
- [14] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton. Software Rejuvenation: Analysis, Module and Applications. In *25th Symposium on Fault Tolerant Computer Systems*, pages 381–390, June 1995.
- [15] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.
- [16] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer. Failure Data Analysis of LAN of Windows NT Based Computers. In *18th Symposium on Reliable and Distributed Systems, SRDS '99*, pages 178–187, 1999.
- [17] B. Murphy and T. Gent. Measuring System and Software Reliability using an Automated Data Collection Process. *Quality and Reliability Engineering International*, 11:341–353, 1995.
- [18] B. Murphy and B. Levidow. Windows 2000 Dependability. In *IEEE International Conference on Dependable Systems and Networks (DSN)*, June 2000.
- [19] S. Ross. *A First Course in Probability*. Prentice Hall, 2002.
- [20] SiliconValley.internet.com. Ebay Outage Twice This Week. <http://siliconvalley.internet.com/news/article/0,,3531-435741,00.html>, Aug. 2000.
- [21] N. Talagala and D. Patterson. An Analysis of Error Behaviour in a Large Storage System. Technical Report UCB/CSD-99-1042, University of California, Berkeley, Computer Science Division, 1999.
- [22] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi. Analysis of Software Rejuvenation in Cluster Systems. In *Joint International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS 2001)*, June 2001.
- [23] K. Vaidyanathan and K. S. Trivedi. A Measurement-Based Model for Estimation of Software Aging in Operational Software Systems. In *International Symposium on Software Reliability Engineering, ISSRE 1999*, November 1999.
- [24] R. V. White. An Introduction to Six Sigma With a Design Example. In *Seventh Annual Applied Power Electronics Conference and Exposition (APEC '92)*, Feb. 1992.
- [25] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Networked Windows NT System Field Failure Data Analysis. In *1999 Pacific Rim International Symposium on Dependable Computing*, Dec. 1999.