Failure Analysis of Internet Services

Archana Ganapathi
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley.

1. Introduction

In the post-PC era, research on reliable Internet Services (IS) is gaining increased attention. The proliferation of networks and information technology places an enormous demand on IS. In particular, the 24 hours a day, 7 days a week delivery of service to customers requires dependable availability. Failures are a fact and recovery/repair is the only solution [16].

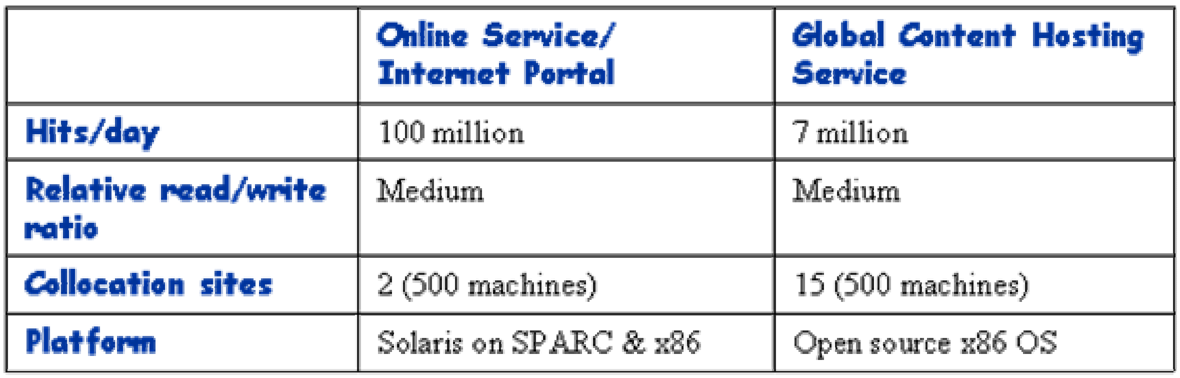
Un-availability can be quantitatively defined as the ratio of MTTR (mean time to repair) to the sum of MTTR (mean time to repair) and MTTF (mean time to failure). Decreasing MTTR is as valuable as increasing MTTF. By learning why systems fail, failure distribution across parts of an IS, we attempt to envision a plausible model for future Information Technology (IT). In this endeavor, we focus on discovering and characterizing failure causes, and suggesting methods to improve recovery from failures, consequently increasing availability. As an example, pervasive use of redundancy improves disaster tolerance. Improving recovery from failures automatically improves availability.

The size, complexity and use of IS often necessitate a globally distributed communications infrastructure. Most IS contain a significantly large number of nodes that are usually geographically distributed, often clustered in collocation sites. With such architectures, dependability is an enormous challenge. Services may fail in multifaceted ways due to numerous diverse components and interactions between them. An improved understanding of failures and their classification is essential to improve availability. One approach to distinguish failures is to partition them into component failures and service failures. While both are due to faulty components, only service failures are visible to the customers.

Previous research in the area of Fault Tolerant Systems (FTS) has attacked this problem from a different perspective. These approaches focus primarily on hardware redundancy to enhance reliability, e.g., using triple modular redundancy and voter logic. They focus on graceful degradation and redundancy to mask failures thereby improving fault tolerance [4, 18]. The Berkeley/Stanford Recovery-Oriented-Computing Project (ROC) emphasizes recovery from failures rather than failure-avoidance [19]. The major motivation for this approach is the observation that most robust systems continue to occasionally encounter failures due to human operator errors, transient or permanent hardware failure, and software anomalies due to "Heisenbugs" or software aging. Software failures are inevitable and a detailed understanding of the causes of downtime is important [12]. ROC's approach collaborates with industry to obtain real data on failure causes, analyze patterns, develop benchmarks that test old systems and generate new ideas to measure improvement, providing support for humans to operate services [16].

Initially, our primary motivation and goals are to study the operational characteristics and failure data of several large-scale IS. Our focus has been to verify and increase the statistical validity of results presented in [15, 3]. We study how maturity affects the number of failures. We provide case studies and data, and in some cases, broaden the range of metrics. This effort will facilitate better approximation of failure models in order to improve the accuracy of benchmarking. These studies also analyze several failure mitigation techniques that improve availability.

Oppenheimer presents data from three Internet services whose architectures are classified as Online, Readmostly and Content [15]. We analyze the Online and Content services in finer detail. The on-line service/Internet portal is termed Online. The global content hosting service is termed Content. The high traffic read-mostly IS is termed Readmostly. All three systems distribute their servers geographically in collocation facilities. The primary differences between these services are their load, read/write ratio and the use of standard web browsing or special purpose software offered to the customer. See figure 1. The Online service receives



	Online Service/ Internet Portal	Global Content Hosting Service
Hits/day	100 million	7 million
Relative read/write ratio	Medium	Medium
Collocation sites	2 (500 machines)	15 (500 machines)
Platform	Solaris on SPARC & x86	Open source x86 OS

Figure 1: Distinguishing characteristics of Internet Services derived from [15].

approximately 100 million hits per day with a medium relative read/write ratio. It has two collocation sites with about 500 machines running Solaris on SPARC and x86 architectures. The Content service receives approximately 7 million hits per day and also has a medium relative read/write ratio [15]. This service has 15 collocation sites with approximately 500 machines running open-source x86 OS.

Oppenheimer suggests taxonomies for classification of failures such as root cause, immediate trigger and type [15]. We define root cause as the chronologically first event in a sequence of events that leads to a service failure. The immediate trigger is the final event preceding the service failure. The focus of failure fixes has been to reduce problem recurrence. Furthermore, an attempt is made to identify components that are frequently involved in a singleton or cascading service failures. Service failures due to multiple component failures are analyzed to identify and understand recurring patterns that cause such failures. This information can be used to build systems that are immune to such patterns and consequently avoid fault propagation and service failures.

2. Related Research

The explosion and popularity of IS in the past decade has placed an unprecedented demand on the reliability of these round-the-clock services [16]. Although failures have been studied extensively in the past within the context of fault-tolerant computing, large scale IS operations focus on issues such as increased rate of software upgrades. Companies such as Amazon Inc., Google Inc. and Yahoo! Inc. serve both as a repository for data in ubiquitous computing systems as well as a platform for building new global-scale applications and services. Potentially, they can provide fascinating solutions to hard problems. It is now expected that these services offer 100% availability to customers of e-commerce, enterprise applications, on-line services and ISPs. Unfortunately, service outages are frequent and outage costs are high, for example, consider the infamous NASDAQ stock-market outage resulting in several million-dollar loss in revenues [2]. In general, in IS, outage costs are enormous. Apart from the social effects that include negative press repercussion and loss of customers who "click over" to competition, the loss in revenues range from half a million to five million dollars per hour [16]. Contemporaneously, rapid deployment of new software, applications and user-interfaces necessitate the change of operational software while continually providing service. Other important facets of IS include maintainability to reduce the burden on system administrators and evolutionary growth to allow easy system expansion over time without sacrificing availability or maintainability.

Researchers on large-scale IS have concentrated on appropriate architectures for such sites [1, 12]. Several studies are pertinent to causes of failures in various types of computer systems, albeit non-IS specific [5]. Kuhn examined data on failures in Public Switched Telephone Systems [9]. He concluded that human error had a significant customer impact on these failures. Enriquez extended this study, considering blocked-calls metric collected from outage reports [3]. She concluded that human error was responsible for over fifty percent of outages, customer minutes and blocked calls.

Failures in networks of workstations have been studied in [25]. A large number of outages are due to planned maintenance, software installation and configuration. System software and planned maintenance caused the largest amount of total downtime. Several researchers have examined failures in enterprise server environments [10, 11, 23]. Relevant research in the area of system monitoring, diagnosis and configuration include [6, 7, 24].

In the on-going effort to work with industry to obtain real data on failure causes and patterns; [15] surveys three sites and [3] provides a survey of FCC switch failure data. The results presented in this paper are an extension of these efforts. Parts of the results corroborate their findings and the remainder provides additional insights into the analysis, considering data gathered over an extended period.

3. Data collection

Once a failure occurs, the first observation is related to the location of the failure. These locations can be the service front-end, mail-handling nodes, service back-end and network domains for Online and service front-end, client front-ends, service back-end and network domains for Content. Distinct categorization occurs due to differing architectures for these services. Front-end nodes serve as the intermediary between the user and back-end nodes. These servers obtain requests from users, forward them to back-end nodes and in the Online service, deliver processed data from back-end nodes to the user. Back-end nodes serve as data storage units and in the case of the Content service, return data requested by front-end nodes to the user [15]. Client nodes are front-ends that are geographically located at the customer site. See figure 2.

The next area of concentration is the cause for the failure: hardware, software, operator, overload, disk-full and environment. Hardware and software errors occur due to faulty or broken hardware and buggy software respectively. Operator errors are caused by human intervention that introduce an error into the existing system. Temporal overloads occur when loads exceed limits that components can comfortably handle without degradation. Disk-fulls(spatial overloads) are attributed to insufficient space on a disk partition.

In our research we have attempted to gather additional data that extends the scope of our initial investigation and consequent classification of failures. The cause field was further classified to elaborate the particular component that caused the failure. For example, hardware is classified into memory and CPU. Similarly, operator error is classified into configuration and procedural categories.

In the failure analysis of US Public Switched Telephone Network (PSTN), Enriquez suggests that it is beneficial to classify human errors into the following categories: vendors, contractors, technicians and outsiders. In our research, instead of classification based on personnel roles, we elaborate human error based on the nature of

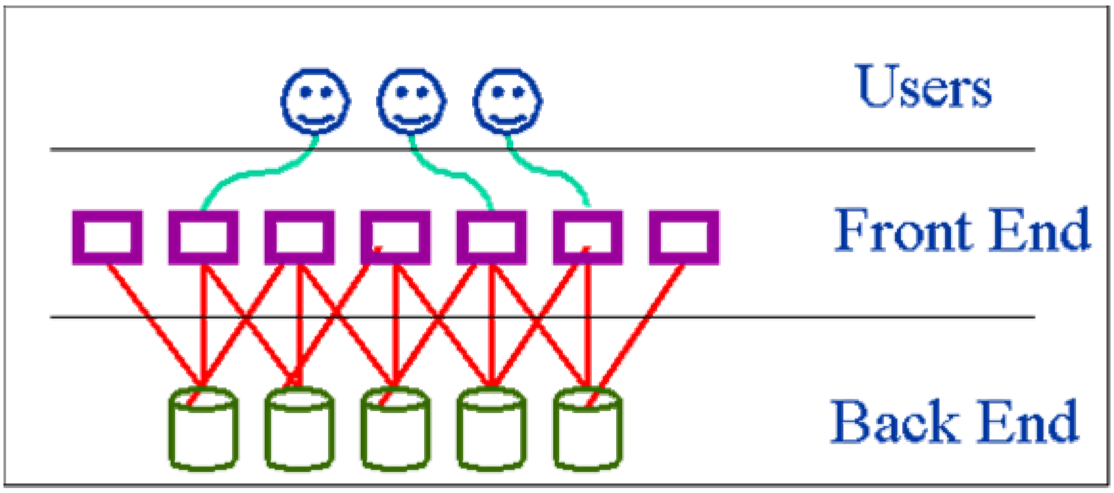


Figure 2: Internet Service Architecture. Client nodes are generalized as front end nodes as only the Content service makes a distinction between the two.

action performed by the operator to cause the failure. Our classification includes two types of operator errors: configuration error and procedural error. For example, a configuration error occurs when the system is configured wrong or in general when the configuration file contains erroneous data. Similarly, procedural errors occur when the operator performs a careless act such as replacing the wrong disk in a Raid5 system. Furthermore, these errors can be introduced during various stages of a task: initial stage, during an upgrade, or when fixing a discovered error. Upon unloading a computer package, the nascent start-up of a system can be erroneous because of the operator's mis-configuration. Similarly, while upgrading software, the operator may be oblivious of relevant changes that are dependent on this upgrade. This oversight results in a configuration or procedural error during upgrade.

We extend the research presented in [15] and [3], incorporating additional data obtained from Online and Content services, probing further in their analysis. Root cause and immediate trigger do not always represent the complete scenario of failures. Often, several events lead to failure. In some cases, each event can be considered as a link in a chain in which a missing link can avoid failure. This chain starts with the earliest occurring event, the root cause, and terminates with the last event, the immediate trigger that caused the failure. The presence of interim links in the chain necessitates the consideration of cascaded failures. Alternately, it is possible to have multiple independent events that do not form a linked chain. However, they cumulatively contribute to a single failure. We show that such failures are more common in Content than Online. We distinguish between these two types of multiple-event failures as *vertically cascaded* (VC) and *horizontally related*(HR) respectively.

4. Analysis

Data was analyzed in multiple passes. The initial classification presented in [15] seemed sufficient at first but as we increased the quantity of data collected; it was more appropriate to extend the initial classification. [15] distinguishes between a component failure and a service failure. A component failure does not impact a customer; however, it carries the potential for a future impact. In contrast, a service failure has an immediate impact on the customer. In our study, customer impacts are further classified into two categories. *PartOfCustomersAffected* (PCA) depicts the part of an entire customer base to which a service failure was visible. *PartOfServiceAffected* (PSA) denotes the effect of service failure observable in operations such as read, write or both for Content IS and similarly, mail, chat or news operations for Online IS.

As discovered by Oppenheimer, there are several causes of failure of which human error is a leading contributor. Our data complies with [15] in this aspect. As visible in figure 3, Operator errors are also the leading cause for high TTR. These errors affected both reading from the site and writing operations to the site in the content-hosting service. In the Online service, it had various repercussions such as degrading performance or inability to use services such as chat and e-mail.

A. Content

Our observation reveals that multiple-event failures are a significant part of operator-induced failures in the Content category of IS. HR failure comprises almost 8% of incidents involving component failures and 25% of incidents involving service failures. Of the HR failures, all affected read operations; most of them (about 88%) affected write operations as well. These statistics were retrieved by counting each service failure once, regardless of the number of component failures causing this service failure. For example, to avoid multiple counts of the same failure, a service failure that had three HR component failures was counted only once despite multiple causes. The chronologically earliest component failure was counted as the cause for the corresponding service failure. All HR service failures had their chronologically first component failure in the client node. Subsequent component failures were also located in client nodes and were mostly software errors.

Service front-ends are responsible for more problems than service back-ends and client nodes. However, 64% of client component failures become service failures while only 46% of network, 15% of front-end, and 12% of back-end component failures become service failures. Client component failures occur at an enormously high rate because the relevant nodes are at the client site and are administered by both the Content service as well as the client service.

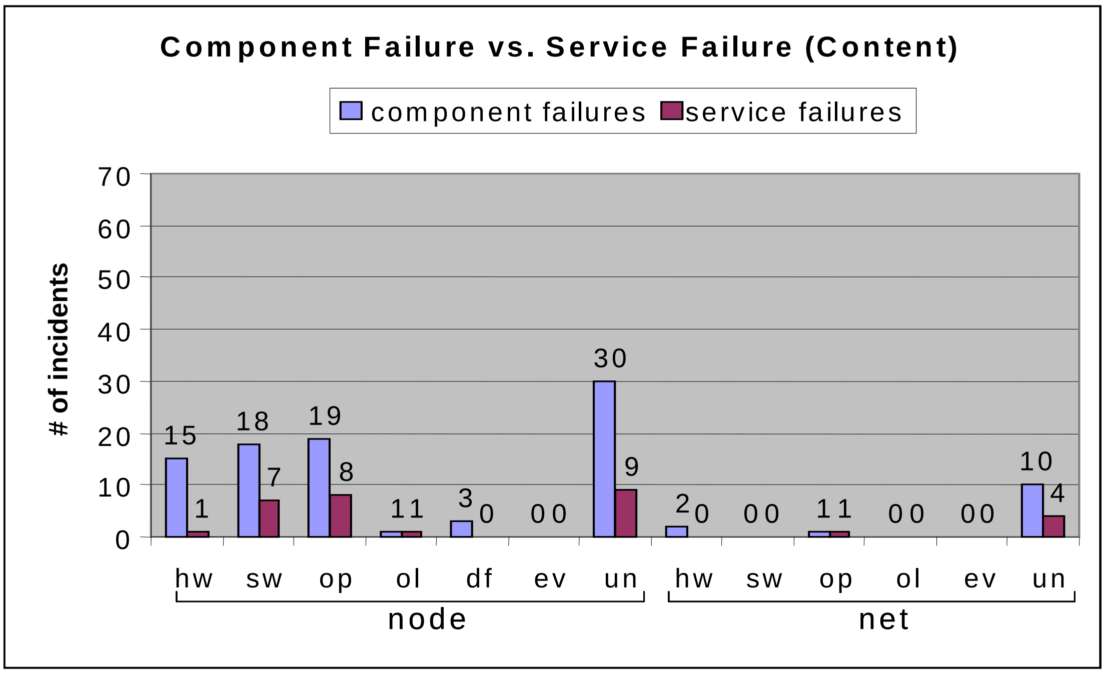


Figure 3: Ratio of component to service failures (Content) Note: These graphs are based on three months of failure data from the Online Service. Node data includes Fe and Be.

There is considerable scope for disconcerted inconsistency in node handling by both services. Network failures can primarily be attributed to the geographical distribution of the service as well as the management of links between the customer and service sites. Front-end problems result in fewer minutes of unavailability. In contrast, back-end failures are significant, albeit infrequent. Unfortunately, operator error is the most difficult to mask. The TTR tables in figure 5 reveal that the back-end average TTR is longest. Client TTR is also high because of issues that remain hard to detect; usually, these issues are related to component failures that occur at a remote site. Often, these components are maintained by administrators who are not comfortably familiar with the equipment; they work for the client company, not the content service. Network service failures achieve the third highest average TTR. Operator errors take the longest time to repair because they are often subtle and hard to discover. For example, a typo in the configuration file is often easily overlooked and thus takes longer to detect. Among these operator errors, procedural errors have the highest TTR as they can range from a knocked off cable to an erroneously replaced RAID disk. Following operator-induced service failures, software errors are predominant. This dominance can be attributed to the fact that programmers write imperfect code. Also, several disk errors appear due to conditions arising from exhaustive use of space in a file-system. Other errors are due to hardware and occasionally due to network break-ins and viruses; however, security issues were not accounted in these calculations.

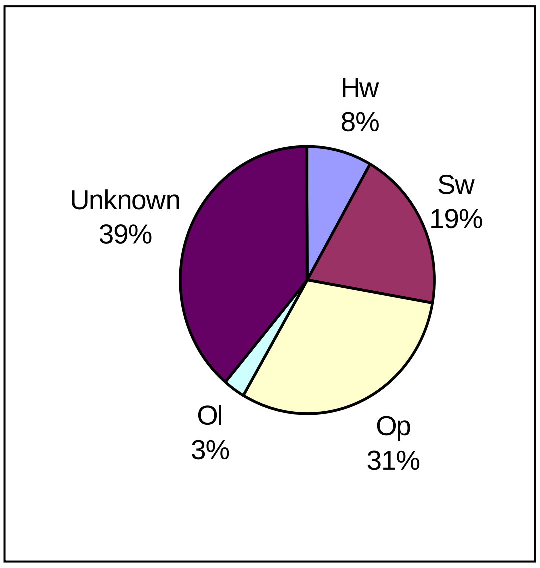


Figure 4: Service failure cause by component (Content). This data was obtained from the analysis of two months of failure data Content hosting service.

	Hw		Sw		Op		Ol		Un	
	Online	Content	Online	Content	Online	Content	Online	Content	Online	Content
Node	25.8 1.5	109 .1	21.4 20.7 2 .5	9.6 2.7 2 .8 .6 .5	50.9 48 6.9	72.9 28 2.9 1.4	47.8 31.8	N/A	8.4 5.3	27.8 6.3 1.3 .5
Net	2.5 1.2	N/A	N/A	N/A	N/A	.2	6.4	N/A	N/A	15.3 .8 .6 .3

Figure 5: Time to Repair in hours for each service failure analyzed in Content and Online Times highlighted in red denote large TTR values. Failures with unusually high TTR due to lower priority of a collocation site (usually a test site) was not included in this table. These numbers could not be retrieved for 16 service failures due to lack of sufficient information in problem reports.

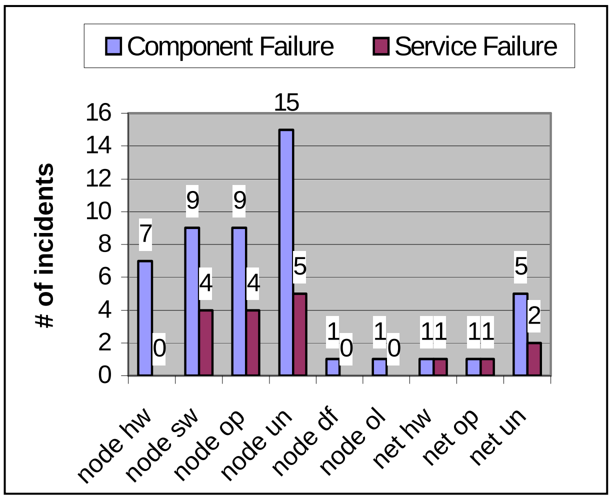


Figure 6: Component vs Service failure (Content) This graph depicts the number of incidents per month based on two months of data compiled from failure reports.

There is a distinction in data between the results presented in [15] and ours. The number of component failures in two months (analyzed here) is equivalent to the number of component failures in one month analyzed in [15]. This disparity can primarily be attributed to the time period of analysis and the changes in service maturity. Since the time of the original analysis, the IS matured from a startup to a relatively stable service. The results provided in [15] suggest that a larger proportion of operator component failures in nodes turn into service failures compared to that of software component failures in nodes. While [15] classifies front-end and back-end machines as nodes, we include client nodes in this category. We analyzed two months of data and computed an average to compare equivalent time periods (one month) of data with [15]. See figures 6 and 7. Our data

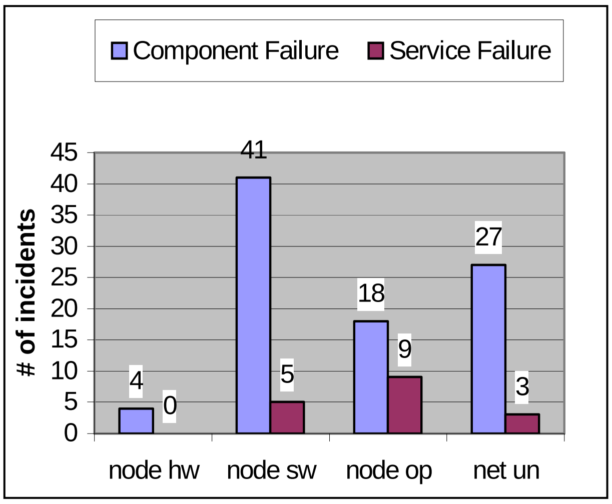


Figure 7: Component vs Service failure (Oppenheimer's Result) for Content This data was retrieved from one month of failure data. Node = Fe+Be+Client

reveals that the proportion of operator component failures in nodes resulting in service failures is almost equivalent to that of software component failures. Moreover, the proliferation of software component failures in our data is less than 50% of similar data in [15]. This discrepancy is potentially due to the fact that software component failures are more effectively masked as a result of service maturity since the time period of data presented by [15]. Conforming to the data from [15], very few hardware component failures that occur in nodes become service failures. Comparatively less network errors appear in our observation than in [15]. Furthermore, the number of node related operator failures has been reduced by almost 50% (this percentage includes both component as well as service failures). Perhaps operators introduce less errors as they become more familiar with relevant components and procedures.

B. Online

We analyzed three months of data from the Online service immediately succeeding the months analyzed in [15]. The data obtained was similar in nature with few differences in the number of service failures caused by each component. These differences can be attributed to the fact that there are few service failures compared to component failures; even a difference of 2 service failures caused by a component can skew percentages.

In Online, service front-ends are responsible for most of the problems. They comprise 84% of all component failures and 77% of all service failures. Contrary to the Content service, network components are the next common cause for both component and service failures. They constitute 12% of all component failures and 18% of service failures. This data closely correlates the Online data in [15]. 9.6% of front-end component failures and 16% of network component failures result in service failures. While back-ends contribute only 3% of component failures, none of these become service failures. Front-end component failures occur at an enormously high rate as there exist numerous individual components in this category that can simply wear out. However, these failures are easily masked due to redundancy in the components and only a small proportion become service failures. Network failures, as in the Content service, are due to the geographical distribution of service.

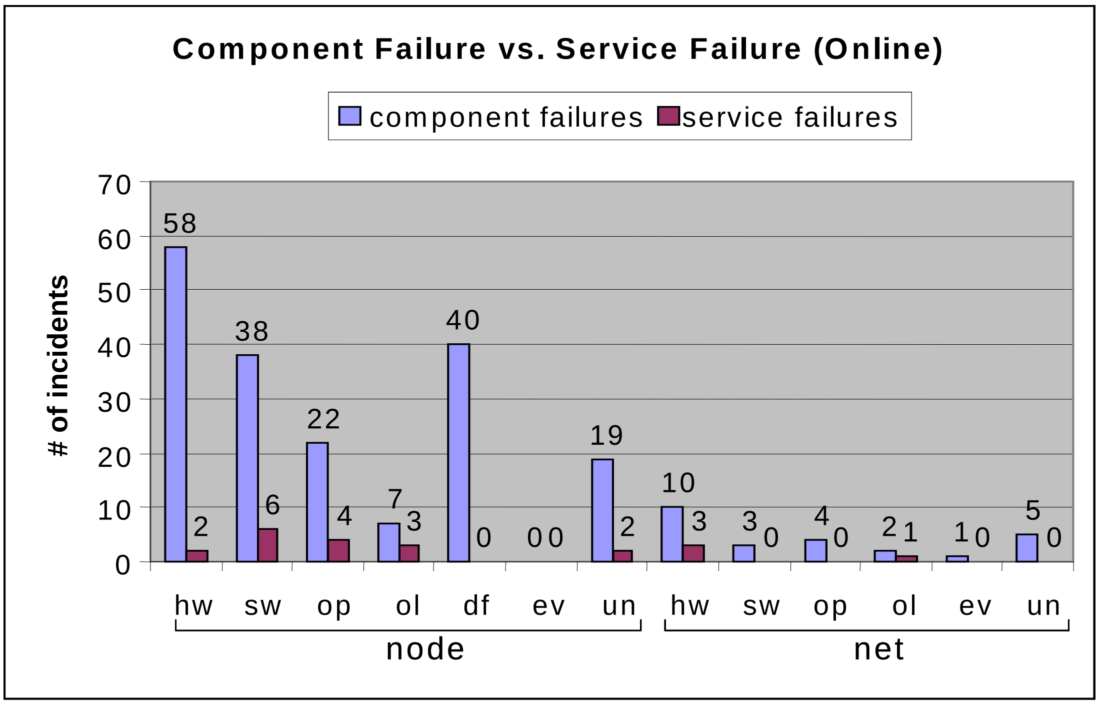


Figure 8: Ratio of component to service failures (Online) These graphs are based on three months of failure data from the Online Service. Node includes Fe and Be.

Nine of the 22 service failures in Online had multiple causes and tended to be vertically cascading. Hardware component failures in the front-end nodes as well as the network were the most common chronologically first cause for service failures. Back-end nodes were the most common chronologically second cause for service failures. Of the 22 service failures, 50% were visible to only part of the customer domain while the remaining 50% were visible to all customers of this service. The distinction lies in the proportion of customers affected (decided by which machine their accounts reside). A particular service was inaccessible to one or few groups of users in the scenario where part of the customers were affected. However, a particular service such as mail or chat was unavailable or noticeably slow to all users regardless of which user group they belonged in the case where all customers were affected. The partial-customer-affecting-service failures occurred in front end nodes and were predominantly due to software component failures. Service failures affecting all customers were primarily due to hardware component failures and often cascaded into unknown back-end component failures, compositely resulting in service failures.

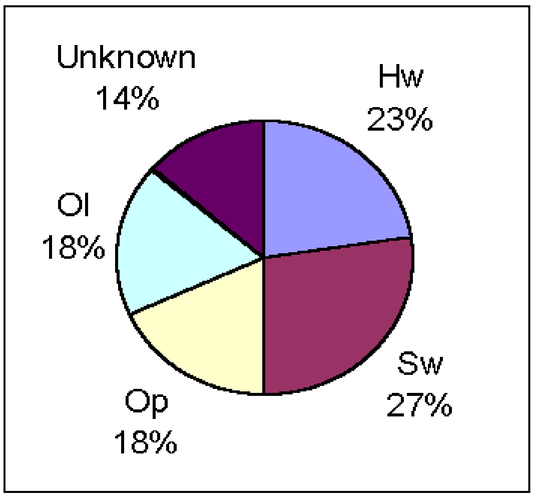


Figure 9: Service failure cause by component (Online). This data was retrieved from 3 months of failure data from the Online Service.

Our Online data also reveals that front-ends have the highest average TTR. Service failures due to network problems have the next highest average TTR and back-ends appear to have the least TTR. This fact contradicts data in [15] as well as the Content data as both of these suggest that back-ends have the highest average TTR (see

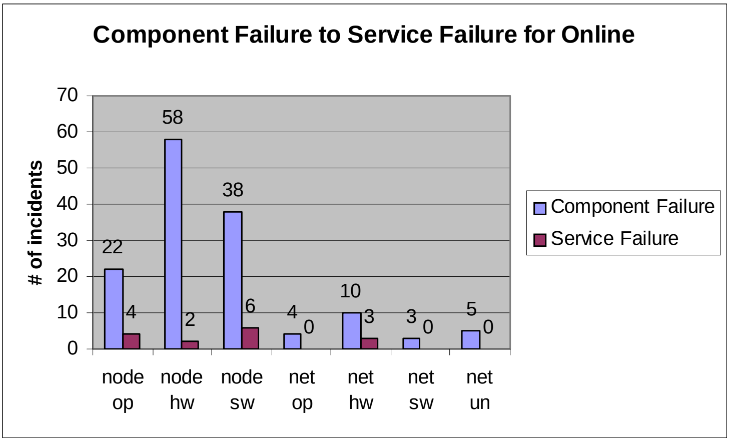


Figure 10: Results from study of Online Service. This data was compiled from 3 months of failure data. Node = Fe+Be

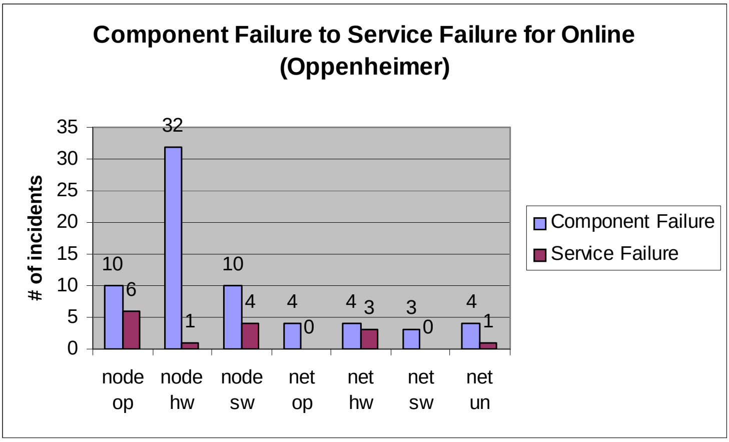


Figure 11: Results from Oppenheimer's study. This data was compiled from four months of failure data.
Node = Fe+Be

figure 5). This discrepancy can be attributed to unusually low number of failures in back-end nodes during the analyzed time period. Among various components used to classify data, overloads resulted in the longest TTR. Perhaps this conclusion is due to the fact that only two service failures were used in this calculation, thus extremely lowering their statistical validity. However, concurring with [15] and the Content data, operator errors have a very high average TTR. The justification for this trend is that these types of errors are harder to detect and/or mask, thus prolonging the process of fixing the error. However, differing from our Content data, operator error, though it had a high TTR, was not the most common cause for service failure. Operator-induced service failures were only the third most common cause of service failure.

Failures due to software bugs were the most common type of service failures. This disagreement between the two internet services can be attributed to the fact that the online service offers more applications to customers and thus contains significantly additional software to maintain. Since software is often dispatched with several bugs, it is not surprising that this cause is most common for service failures in the Online service. The second most common cause for service failures was hardware. However, the totals for all three of these categories were off by only one service failure (6 software, 5 hardware, 4 operator). Perhaps with additional data, the rifts between these numbers can widen and expose additional insight into the common cause for failure.

Compared to [15], several more hardware, software and operator-induced component failures occur in nodes. Our data reveals more than twice as many operator and software component failures and a little less than twice as many hardware component failures. However, the number of service failures in Online service in each of these components remains similar to data analyzed in [15]. Consequently, an overall smaller percentage of component failures become service failures in our data. The increased number of component failures can be due to an increase in the number of individual components used by the service. However, it appears as though these new component failures are masked fairly effectively to avoid additional service failures.

C. Techniques for Failure Mitigation

IS failures can have multiple causes: root cause, immediate trigger and numerous intermediate component failures. Perhaps a useful outage metric is the product of outage duration and the number of customers affected by the outage. We have identified failure root causes and their mean time to repair. However, this metric only provides the time taken for the problem to be fixed from the time it occurred. Often, a service failure is user-visible only for a small interval and may continue to be a component failure for a longer period of time. Early detection of failure and redundancy reduce performance degradation and can effectively minimize the time to repair.

The normal fault propagation path is:
Component failure → Service failure → degradation in user perceived Quality of Service (QoS).

The following techniques help mitigate failures and sometimes prove beneficial in masking them:

- Testing (pre-deployment as well as online)
- Redundancy (replicate data, computational functionality, networking functionality)
- Automatic sanity checks of configuration files
- Fault and load injection (pre-deployed fault injection and load testing supporting the notion of 'prevention better than cure' as well as online testing to imitate excessive loads after deployment)
- Increased isolation between software components to prevent failure propagation
- Periodic prophylactic restarts to avoid latent errors (e.g., memory leaks, available swap space)

Of the failures that were analyzed, about 90% show potential for mitigation by ROC techniques. In Content, Configuration check appears to be the most useful ROC technique,

Pre test	Pre flt/ld	Online test	Online flt/ld	Red	Config	Re-start	Exp/ mon ttd	Exp/ mon ttr	None	N/A
5	5	8	10	5	15	2	7	2	5	14

Figure 12: Number of service failures that would have been mitigated by applying ROC techniques. A total of 52 service failures were analyzed. Red represents redundancy; flt/ld stands for fault/load injection; config signifies configuration checking; isol stands for isolation of components; Pre test represents pre-deployment testing; Exp/mon TTD suggests better exposure/monitoring tools to decrease time to discovery while Exp/mon TTR suggests better exposure/monitoring tools to decrease time to repair.

followed by Online fault/load injection. This result may be attributed to the fact that most operator errors appear in configurations. Online testing as well as better exposure/monitoring to reduce failure discovery time appear to be the most useful technique in the Online service followed by Online fault/load injection and Configuration checking. These results are not surprising as the service itself is an online service and continuous online testing can reveal problems and provide opportunities for faster repair. The combined total for both these services reveal that Configuration checking is the single most useful method for mitigating failures. Closely following this mitigation technique are online testing and online fault/load injection. Perhaps these techniques alone would reduce the number of failures by at least one half. These services already isolated components effectively so this technique would not help further mitigation. Also, it is evident that once the failing component was correctly discovered, it was efficiently repaired. Thus, simply improving exposure of failures would reduce the total time a failure is user-visible.

D. Case Studies

There were several interesting service failures encountered during the data collection process. Some multiple cause failures are exemplified below.

In a service failure in Online, the ACL in a front-end mail-handling machine was mis-configured and consequently incoming mail from two particular e-mail web servers was blocked. Once the problem was discovered and fixed, all queued mail came pouring in. In turn, this delay caused front-end machines to be overloaded as they were unable to handle the amount of legitimate incoming mail. As the first component failure directly caused the second, this failure is an example of a vertically cascaded failure. It took considerable time for the mail system to stabilize. This problem can be mitigated by automatic configuration checks so that all mail-handling machines can check against a standard set of configurations issuing warnings if any discrepancies existed.

‘Contact lost’ alarms were noticed on various machines on the production DB network at the Online service. The router between the db network and the remainder of the service was problematic. This problem caused a high load average on a main back-end db machine. Thus, they rolled over to another back-end db machine, which also had a high load average in due course. Eventually, the entire db network was unavailable so the service failed and the SOC decided to switch over to a new router. They manually switched cables from one router to another. This problem was vertically cascading as one component failure lead to another and the failures propagated through a chain resulting eventually in a service failure. Perhaps better exposure of the various components and monitoring them can improve the situation. If the router was monitored more closely, it will take less time to detect and correct this problem. Also, load testing will revealed that the back-end machines cannot handle this overload and the administrators can consider options to alleviate the load.

In the Content service, an application failed a client box. Under normal circumstances, a failover would occur and the other box in the pair would take over. However, this box also had problems and thus the failover occurred much later. The first component failure did not cause the delay in failover. However, an independent component failure in the second box coupled with the first component failure due to the application in the first box resulted in a service failure affecting both reading and writing abilities from the client site to the Content service. Thus, this problem is horizontally related. Perhaps more redundancy in client boxes can alleviate this situation so that both boxes can failover to a third and fourth box.

At a client site in the Content service, one machine in a pair of client machines was dysfunctional. The other machine had five different configuration errors that were mis-configurations performed by the Content service at the time of machine dispatch. Consequently the customer was unable to ‘mount’. As both boxes in the pair were unavailable simultaneously the services were unavailable. This problem is also horizontally related as the two component failures occurred independent of each other. Eventually, these configurations were fixed and the machines were operational. Checking configurations would have definitely mitigated this failure. Also, increased redundancy in client boxes can help this scenario as the fail-over can be targeted to a third box which was functional at that time.

5. Conclusion

From this investigation, it is safe to conclude that operator-induced errors are most impacting. Human intervention causes a significant number of service failures and is also the hardest failure to mask. The time to discover and repair operator errors is much greater than that for software and hardware failures. The most valuable failure mitigation techniques are configuration checking, online testing and online fault/load injection. These techniques carry the potential to efficiently reveal plausible service failures and in many cases, prevent their occurrence in Internet Services.

6. Future Research Directions

Many enhancements can be made to this study. Oppenheimer has suggested several ideas in his thesis [15]. I have elaborated on those ideas as well as mentioned some of my own. Firstly, we can develop a classification method that effectively incorporates Vertically Cascading and Horizontally Related service failures. Perhaps this information will reveal trends in the data due to multiple-event failures. Similar trends can be revealed if we develop failure models based on the time of day when the failure occurred. There exists significant scope to explore failure mitigation techniques and relate this study with newer classifications based on multiple-event failures. Eventually, this information and statistics can be applied towards developing accurate models for benchmarking. The remaining categories are in automated compiling systems that help produce *dependable systems* as well as research on the architecture of Internet Systems. I have categorized and elaborated these research directions in the following sub-sections.

A. Failure data-collection and analysis

A large and thorough study of IS failures embellished by anthropological fieldwork is a stepping stone in corroborating the statistical validity of our results. The first step is to broaden the range of quantitative metrics and account for degradation in Quality of Service (QoS). This study must include an investigation of the effect these failures have on customers and also perform cost accounting of lost revenue. The role of failure propagation in causing service failures, horizontal/vertical cascade, as discovered in my research will help correlate failures to their impact on QoS. In this endeavor, it is beneficial to examine data from a broader spectrum of services covering various Internet Service architectures. Furthermore, certain key issues remain uninvestigated. Specifically, my research did not address issues such as TTD (time to detection), which is an important contributor to "availability". This research can potentially result in additional taxonomies for classification.

B. Failure Mitigation Techniques

After accruing sufficient data to analyze failures, the next step is to investigate how to mitigate service failures. Although [15] has developed a useful model for failure escalation, from fault to failure to detection to repair, cascaded errors are especially interesting and deserve further attention. Monitoring can play a significant role in predicting component failure, thus engaging procedures that avert component failure. Perhaps architectural support can improve software reliability [14]. It is attractive when compared to the expensive approach of having software monitor its own execution and recover from them. Hardware support has been used in the past for performance monitoring whereas now we are interested in improving the reliability of ISAs. More generally, a computing utility system architecture with "compute capsules" on a globally distributed collection of anonymous computing resources can help facilitate non-homogeneous monitoring in the IS environment [22].

The virtues and consequent effect of periodic prophylactic rebooting and monitoring on cascaded failures must also be considered. It is expected that people who develop, operate and rely upon Internet services will particularly benefit from this study, as will a larger audience interested in system failure in a broader sense. Mitigation techniques such as pre-dispatch testing, online testing, configuration checking and prophylactic restarts must be considered. The next step is to quantify their individual contributions. The primary medium is error injection and measuring the consequent impact of failures and service availability. Metrics such as throughput, fraction of available service functionality, response time, and financial cost of unavailability must be used in quantification.

C. Synthesize Benchmarks from Failure data

Next, it is fruitful to design an accurate model for benchmarking. Internet Service benchmarks are more difficult to design than those for traditional services. Mostly, they comprise a mixture of high-level tasks rather than simple protocol-level workloads. One significant research direction would be to design a stochastic fault-model, based on data already gathered, to drive service-level dependability and recoverability. Benchmarking recoverability is based on QoS, whose decrease is an inverse measure of recoverability. Benchmarking dependability incorporates the mean time to failure of each failing component. Perhaps a beneficial metric comprises weighting QoS responses to recovery events by the frequencies of component failures and their impact based on their mean time to recovery.

D. Research on Internet Services Architecture (ISA)

Another important direction is in the area of Internet Services Architecture. As a side effect of this research, one can answer questions related to the design and analysis of the architecture of IS. Some examples are the following questions. *What is the effective architecture for current Internet services?* Currently our discussion is quite limited to a vague notion of stateless front ends, and stateful back ends. *What exactly is "business logic" and how is it embodied in the service? How is state synchronized across multiple collocation facilities? Why is so much custom software required? Is there a means to abstract internal hosting center topology?* It will improve our understanding of how various IS services differ from one another. *What are particular challenges faced by one service relative to another?*

The above research directions cumulatively provide dependable performance in Internet Services by effectively decreasing the number of customer-impacting failures. Tools designed as parts of this endeavor will not only provide feedback to prevent component degeneration but also yield results to improve the service model and most importantly, availability. In addition I expect this research endeavor to improve the design and architecture of Internet Systems. Most importantly, these efforts will contribute to the advancement of Dependable Internet Systems.

7. Acknowledgements

I wish to thank several people for their constructive criticisms and tireless efforts to edit this paper. I would like to thank Professor David Patterson for his continual guidance and feedback throughout this investigation. I would also like to thank David Oppenheimer for helping at every stage of the data analysis and providing valuable feedback on my work as well as this paper. Last but not least, I would like to thank everyone who proof-read this paper.

8. References

[1] Brewer E. Lessons from giant-scale services. *IEEE Internet Computing*, July 2001.

[2] Cable News Network Television. http://money.cnn.com/1999/10/06/markets/nasdaq/index.htm

[3] Enriquez P, Brown A. and Patterson D.A. Lessons from the PSTN for dependable computing submission to Workshop on Self Healing, Adaptive and self-MAN-aged systems, 2002.

[4] Chandra S. and Chen P.M. How Fail-Stop are Faulty Programs? *Proc. of the 1998 Symp .on Fault-Tolerant Computing (FTCS)*, June 1998.

[5] Gray J. Why do computers stop and what can be done about it? *Symposium on reliability in distributed software and database systems*, 3-12, 1986.

[6] Hewlett Packard. HP OpenView http://www.openview.hp.com/

[7] IBM Tivoli software. http://www.tivoli.com/

[8] Spector, A. IBM. Autonomic Computing, ROC Retreat, Tahoe City, CA, June 10-12, 2002.

[9] Kuhn D.R. Sources of failure in the public switched telephone network. *IEEE Computer* 30(4), 1997.

[10] Lancaster L. and Rowe A. Measuring real-world data availability. *Proceedings of LISA 2001*, 2001.

[11] Lee I. and Iyer R. Software dependability in the Tandem GUARDIAN system. *IEEE*

Transactions on Software Engineering, 21(5), 1995.

[12] Microsoft TechNet. Building scalable services.http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/itsolutions/ecommerce/deploy/proj-plan/bss1.asp, 2001.

[13] Hamilton, J. Microsoft. Active Server Availability. Feedback, ROC Retreat, Tahoe City, CA, June 10-12, 2002.

[14] Oplinger, J. & Lam, M.S. Enhancing Software Reliability using Speculative Threads, In Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems, October 2002.

[15] Oppenheimer D. Why do Internet Services fail, and what can be done about it? MS thesis, Computer Science Division University of California Berkeley, May 2002.

[16] Patterson D.A. Recovery-Oriented Computing. ROC Retreat, Winter 2002 Lake Tahoe, Nevada, January 2002.

[17] Patterson, D. A. A simple way to estimate the cost of downtime. Submission to 16th Systems Administration Conference (LISA'02), 2002.

[18] Patterson, D.A., Gibson, G. A. and Katz, R.H. A case for redundant arrays of inexpensive disks (RAID). Technical Report UCB/CSD 87/391 University of California, Berkeley. also appeared in ACM SIGMOD Conf. Proc., Chicago, June 1-3, 1988, 109-116

[19] Patterson, D. A., A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A.Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W.Tetzlaff, J. Traupman, N. Treuhaft. Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. UC Berkeley Computer Science Technical Report UCB//CSD-02-1175, March 15, 2002.

[20] Brown, A. and Patterson, D.A. Rewind, Repair, Replay: Three R's to Dependability. To appear in 10th ACM SIGOPS European Workshop, Saint-Emilion, France, September 2002.

[21] Oppenheimer, D., A. Brown, J. Beck, D. Hettena, J. Kuroda, N. Treuhaft, D.A. Patterson, and K. Yelick. ROC-1: Hardware Support for Recovery-Oriented Computing. IEEE Transactions on Computers, vol. 51, no. 2, February 2002.

[22] Schmidt, B.K. Supporting Ubiquitous Computing with Stateless Consoles and Computation Caches Ph.D. Thesis, Computer Science Department, Stanford University, August 2000.

[23] Sullivan M. S. and Chillarege R. A comparison of software defects in database management systems and operating systems. In *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, 1992.

[24] Sun Microsystems. Solaris JumpStart. http://wwws.sun.com/software/solaris/archive/8/ds/ds-webstart/

[25] Thakur A. and Iyer R. Analyze-NOW-an environment for collection and analysis of failures in a network of workstations. *IEEE Transactions on Reliability*, R46 (4), 1996.

[26] Xu J., Kalbarczyk Z. and Iyer R. Networked Windows NT System Field Failure Data Analysis. http://www.crhc.uiuc.edu/~junxu/resume/Papers/PRDC99_camera_ready.pdf

Appendix A – Content Data Tables

Note: CF = component failure, SF = service failure

A. Characteristics of Failure Data

	Faults	**Failures**	**Time period**
Content	106	36	2 months

B. Component/Service Failure cause by location

	Fe	**Be**	**Net**	**Client**	**Cust-net**	**Cust-unk**	**Unk**	**Total**
Content(CF)	41(38.7%)	17(16%)	13(12.3%)	28(26.4%)	2(1.9%)	1(.9%)	4(3.8%)	106
Content(SF)	6(16.7%)	2(5.6%)	6(16.7%)	18(50%)	2(5.6%)	0	2(5.6%)	36

C. Chronologically First CF/SF cause by component and type of cause (Fe) ---All failures impacted both read and write

	Hw Disk	**Sw App**	**Sw Apph**	**Hw Swx**	**Op Con**	**Op Proc**	**Un**	**Tot**
Content (CF)	3	8	2	1	9	1	17	41
Content (SF)	0	1	1	0	2	0	2	6

D. Chronologically Second CF/SF cause by component and type of cause (Fe) ---All failures impacted both read and write

	Sw Apph	**Op Con**	**Df**	**Tot**
Content (CF)	1	1	1	3
Content (SF)	0	1	0	1

E. Chronologically First CF/SF cause by component and type of cause (Be) ---- (R) = affected read, (B) = affected both read and write

	Hw Disk	**Sw App**	**Op Proc**	**Df**	**Un**	**Tot**
Content (CF)	10	1	1	3	2	17
Content (SF)	1 (R)	0	1(B)	0	0	2

F. Chronologically First CF/SF cause by component and type of cause (Net) ---- All failures impacted both read and write

	Hw Cable	**Hw Swx**	**Op Config**	**Unk Wan**	**Unk Lan**	**Un Router**	**Unk**	**Tot**
Content (CF)	1	1	1	1	1	2	6	13
Content (SF)	0	1	1	0	0	2	2	6

G. Chronologically First CF/SF cause by component and type of cause (Client) ---- (R) = affected read, (W) = affected write, (B) = affected read and write

	Hw Disk	**Sw App**	**Op ConUp**	**Op ConIn**	**Op ConUn**	**Op Other**	**Ol**	**Un**	**Tot**
Content	1	7	1	2	4	1	1	11	28

(CF)									
Content (SF)	0	5(B)	1(B)	1(B)	2(B)	1(B)	1(B)	7(5B, R1, W1)	18

H. Chronologically Second CF/SF cause by component and type of cause (Client) ---- (R) = affected read, (W) = affected write, (B) = affected read and write

	Sw App	**Op ConIn**	**Tot**
Content (CF)	6	1	7
Content (SF)	6(5B,1R)	1(B)	7

I. **Chronologically Third CF/SF cause by component and type of cause (Client) ---- 1 Service Failure** (1 Component Failure) – OpConfigInit affected read and write

J. Chronologically First CF/SF cause by component and type of cause (Cust-net) ---- (B) = affected both read and write

	Op ConUp	**Un**	**Tot**
Content (CF)	1	1	2
Content (SF)	1(B)	1(B)	2

K. **Chronologically First CF/SF cause by component and type of cause (Cust-unk) ----** 1 fault, no failures

L. Chronologically First CF/SF cause by component and type of cause (Unk) ---- (R) = affected read, (W) = affected write, (B) = affected read and write

	Hw Mem	**Op Con**	**Tot**
Content (CF)	3	1	4
Content (SF)	1(R)	1(W)	2

M. Number of failures due to fault in root cause components listed below:

	Hw	**Sw**	**Op**	**Ol**	**UnRouter**	**Un**	**Total**
Content	3(8.3%)	7(19.4%)	11(30.6%)	1(2.8%)	2(5.6%)	12(33.3%)	36

Appendix B – Online Data Tables

Note: CF = component failure, SF = service failure

A. Characteristics of Failure Data

	Component Failures	Service Failures	Time period
Online	210	22	3 months

B. Component/Service Failure cause by location

	Fe	Be	Net	Unk	Total
Online(CF)	177(84.3%)	7(3.3%)	25(11.9%)	1(0.5%)	210
Online(SF)	17(77.3%)	0	4(18.2%)	1(4.5%)	22

C. Chronologically First CF/SF cause by component and type of cause (Fe)

	Hw	Sw App	Op Con	Op Proc	Op Un	Df	Ol	Un	Tot
Online (CF)	55	38	14	2	5	38	7	18	177
Online (SF)	2	6	3	0	1	0	3	2	17

D. Chronologically Second Fault/Failure cause by component and type of cause (Fe)

	Sw Apph	Hw	Ol	Df	Tot
Online (CF)	4	3	6	6	19
Online (SF)	1	0	1	0	2

E. Chronologically Third Fault/Failure cause by component and type of cause (Fe)

2 faults—1 ol and 1 sw; no failures.

F. Chronologically First Fault/Failure cause by component and type of cause (Be)

	Hw	Op Con	Df	Un	Tot
Online (CF)	3	1	2	1	7
Online (SF)	0	0	0	0	0

G. Chronologically Second Fault/Failure cause by component and type of cause (Be)

	Sw	Un	Tot
Online (CF)	1	6	7
Online (SF)	1	4	5

H. Chronologically First Fault/Failure cause by component and type of cause (Net)

	Hw	Sw	Op Con	Op Proc	Ev	Ol	Un	Tot
Online (CF)	10	3	2	2	1	2	5	23
Online (SF)	3	0	0	0	0	1	0	4

I. Chronologically First Fault/Failure cause by component and type of cause (Net)
1 fault (net-un-alteon) no failures

J. Chronologically First Fault/Failure cause by component and type of cause (Unk)
1 fault/1 failure in Unk-un

K. Number of failures due to fault in root cause components listed below:

	Hw	**Sw**	**Op**	**Ol**	**Df**	**Ev**	**Un**	**Total**
Online	5	6	4	4	0	0	3	22