Understanding Large System Failures - A Fault Injection Experiment

Ram Chillarege
Nicholas S. Bowen

IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY-10598

ABSTRACT

This paper uses fault injection to characterize large system failures. Thus, it overcomes limitations imposed by the lack of complete information in field failure data. The experiment is conducted on a commercial transaction processing system and this paper:

- Introduces the idea of *failure acceleration* to conduct such experiments.
- Estimates total loss of the primary service to occurs in only 16% of the faults.
- Reveals errors termed *potential hazards* that do not affect short term availability but cause a catastrophic failure following a change in operating state.
- Identifies at-least 41% of errors as potential candidates for repair before total failure. Estimates are provided on the likely location of such errors in terms of storage area and code-data split.

These results enhance our understanding of large system failures and provide a foundation for design enhancements, and modelling of availability.

Index Terms - Fault Injection Experiments, Software Failure, Large System Availability.

I. INTRODUCTION

Failure in large systems is a complex process that involves many layers of recovery. Understanding this process, or its characteristics, is key to enhancing the design of systems. A traditional technique employed to gain this understanding has been the analysis of field failure data. Although such studies [Mourad87] [Iyer82] have been useful, they are predominantly limited by the type of information contained in field data. For instance, cause and effect analysis is hard and often impossible to ascertain from field data. This paper uses a different approach - fault injection on a commercial transaction processing system, performed in the laboratory. This experimental approach has some unique advantages in its ability to make measurements and focus on specific problems of interest.

Figure 1 portrays the domain of an experiment compared to that of field. A machine in a good state is put into the error state by a fault. Usage of erroneous information leads to failure. However, not all errors lead to failure; an error caused by a transient fault may be overwritten, returning the machine to the good state. Errors can also remain latent for a very large time [Chillarege87] and never noticed, for instance, when a scheduled IPL (restart) forestalls the failure. The domain of field failure data is limited to what is identified and reported in the failure logs, i.e., transitions to the failed state. Transitions from the error state which do not cause a failure are outside the domain of field data, but remain within the domain of an experiment. Furthermore, since the cause of failure (fault) is controlled and the effect measurable, the experiment provides greater opportunity for understanding the related phenomenon.

This paper presents a methodology and results from an experiment conducted to study failure characteristics of a commercial transaction processing system. [Hummel88] [Segall88], are two recent fault injection studies. Both emphasize automated fault injection testing and are geared to conduct a large number of fault injections. We cannot attempt a reasonable comparison with our results, since neither study is on a comparable system. In our experiments we do not attempt any automation. Our focus is on the underlying philosophy for constructing fault injection experiments and what we learn from them. Section II is devoted to the design methodology for such experiments and the introduction of *failure acceleration*. While failure acceleration has general applicability to the design of such experiments, the bulk of this paper is directed towards the specific results from our experiment. Section III analyzes the failures which resulted from the experiments. These are viewed as they affect the *primary service* and consequently the availability of the data center. This is followed by a detailed analysis of errors that did not cause an outage. Such analysis has lead to the concept of a *potential hazard* which can explain earlier unexplained

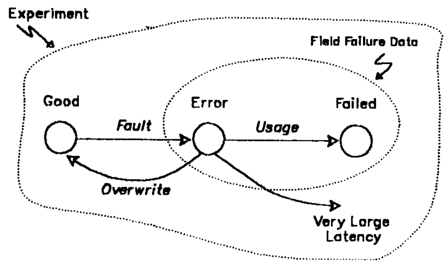

Figure 1. Domain of Experiment and Field Data.

0731-3071/89/0000/0356$01.00 © 1989 IEEE

phenomenon regarding workload and failure. Section IV develops the idea of failure prevention and error repair. We provide specific measures (such as, code or data damage, and storage area distribution) to quantify types of errors which are candidates for failure prevention. Finally, the Conclusions summarize the results of this paper.

II. DESIGN OF THE EXPERIMENT

This section develops the rationale and methodology used to conduct the experiment. We first describe the concept of failure acceleration, followed by the choice of the fault model, the implementation of the fault injection mechanism, and finally the system and workload used for the experiment.

Failure Acceleration

An important attribute of a fault injection experiment is the ability to measure in the lab what could not be done in the field. This includes many aspects of the failure process including, when possible, cause and effect analysis. Interestingly, there seems to be a mechanism that allows for a number of measurements, if a few conditions are met, or if the system can be stressed towards meeting these conditions. These conditions result in accelerating the failure process. We first define it formally and then discuss its attributes.

Definition - Failure Acceleration:
The failure process is said to be accelerated when, the fault model is not altered and:

1. *The fault latency is decreased.*
2. *The error latency is decreased.*
3. *The probability of a fault causing a failure is increased.*

Definition - Maximum Failure Acceleration:
The failure process is said to be maximally accelerated when, the fault model is not altered and:

1. *The fault latency is zero.*
2. *The error latency is a minimum.*
3. *The probability of a fault causing a failure is a maximum.*

It should be carefully noted that the failure process is being accelerated, not the fault occurrence. There is little similarity to the acceleration method used in the semiconductor industry wherein the temperature is raised and the Arrhenius model used to estimate the quiescent failure rate: raising the temperature decreases tolerances and increases the fault rate. In our fault injection experiment, we introduce faults and the rate of fault occurrence has no meaning. It is the process which follows the fault occurrence we study, characterize, and measure. For this reason, the fault inserted in a fault injection experiment needs to be carefully chosen to make meaningful measurements. We first examine what failure acceleration buys and then see how to achieve it.

Failure acceleration increases the speed with which the state transitions occur between the good, error and failed states (reference to Figure 1). Maximum failure acceleration implies that transitions between the good and error states are instantaneous, and the system remains in error for the shortest period of time. A failure accelerated system makes for experiments to be conducted without excessive time spent waiting for consequences of a fault. Our experience (as discussed later) shows that low latency errors are accelerated to fail within seconds. This has the advantage that experiments can be performed, within reasonable time limits, measuring the short term effects with on-line monitors. Errors with very large latency remain latent and techniques such as dump analysis and function identification are needed for analysis. Since acceleration changes the time domain measurements, such as latencies, it cannot be used to make MTBF type measurements. On the contrary, latencies in the field are assured to be higher than in the experiment. This is important to failure prevention techniques that capitalize on large latencies, discussed in section IV.

The other attribute of a maximally accelerated system is the probability of a fault causing a failure is maximized. Thus, the measurements from an experiment tend towards extreme values providing empirical lower and upper bound estimates for transitions to the good and failed states respectively. With reference to Figure 1, transitions to the failed state are maximized, providing an empirical upper bound measurement. Similarly, measurements of transitions from the error state which do not result in the failed state estimate lower bounds. Thus, acceleration provides the unique ability to make some very useful measurements. These bounds can be used to project from field data, measurements that are otherwise not directly possible. For example, the fraction of faults which do not lead to failure (lower bound) is useful to estimate the incident (actual) fault rate from the measured (field) failure rate.

The requirement that the fault model not be altered is important, given that acceleration is achieved by controlling the injected fault and workload. Changing the fault model would imply, for example, a single fault appear as a multiple fault, or a timing fault a control fault. Engineering judgment on the mechanics and effects of an injected fault is critical in designing for acceleration.

Fault Model and the Injected Fault.

This experiment is intended to see the effects of *software faults* particularly, one noted to have a predominance in the industry, the *overlay*. An overlay occurs when a program writes into an area of storage due to an incorrect destination operand. The program causing an overlay has addressability to the area overlayed, but the address to which data is written is in error. An overlay can occur over either data or code. [Endres75] provides an analysis of errors and their causes in system programs. A large fraction of them (approximately 34%) can be mapped into the overlay model.

The fault injection is designed to emulate the effects of a software overlay and accomplish acceleration, preferably the maximum acceleration. This is implemented by choosing a random page of real storage in use and setting its contents to hexadecimal 'FF'. It is a simple fault but accomplishes all the requirements of failure acceleration. We examine them one by one: The hex 'FF' value has been selected because in the 370/XA architecture it is an invalid opcode, an invalid branch location, and generally an invalid address. Thus, if the fault is used it is very likely it will be detected, thereby increasing *the probability of the fault causing a failure*. The fault is only inserted on a page that is currently in use, and the error condition is immediately effective thereby making the *fault latency zero.* (Our experience with this workload indicates a page full of hex 'FF' is not one of the likely legitimate states). The *error latency is minimized* due to a few reasons. Firstly, the experiment is conducted with a workload pushing towards limits in CPU and I/O capacity, reducing error latency [Chillarege87]. Secondly, the real storage does constitute the most frequently used virtual storage in the system and is thereby assured to have the least error latency. Lastly, although virtual storage requests can be in any size, real storage is allocated in quantums of one page. Thus, access within any part of the page will cause a failure. Limiting the fault to one real storage page is done to maintain the requirement that *the fault model should not be altered* by failure acceleration. A single page of real storage is the largest amount of contiguous real storage guaranteed to be owned by a single address space. This ensures each experiment can be treated as a single fault, thereby not altering the fault model. Further, since each run of the experiment uses a random page, equivalent to a single fault, together they constitute a set independent trials.

Critics may claim that the range of software overlays is not the entire virtual storage space. This study does not make this claim. Field failure data, along with reports from software trouble shooting teams suggest there are areas of virtual storage more likely to be the target of an overlay. An argument may claim faults in the Multiple Virtual Storage (MVS) operating system are more likely to occur in common areas having multiple writers. Although this reasoning may be sound, we have taken a different approach. As previously discussed, the domain of errors identified through field failure data cannot be comprehensive. Failure (i.e., an observed error) has been the only measure of fault occurrence. Thus it is not possible to claim the whereabouts of faults. Therefore, this study does not make any assumptions on where software overlays can occur and allows the whole virtual space to be the range of failures. This is accomplished by choosing the location of the fault randomly.

Injecting the Fault

A practical issue in the ability to inject the fault, into a random page in real storage (frame), is the difficulty in addressability to the chosen frame. If a selected real frame is backing a virtual page in the private area of another address space, the program can not directly address the selected page. The solution to the addressability problem is to turn dynamic address translation (DAT) off and write into the page using a real address. When DAT is turned off the logical instruction and data addresses are treated as real addresses. However, for the purpose of analysis we need to know the attributes of the real storage frame such as the address space identifier and associated status information on the frame. For this purpose, we reverse the actions of dynamic address translation to get information on who (in the virtual space) owns the real storage.

The experiment was conducted on a system with 32 Megabytes of real storage. The page frame table for real storage frames consists of an entry for each real storage frame. The address space identifier of the owner is contained in the entry as well as the virtual storage address by which the owner would reference the page. The virtual storage address can be used to determine the area (such as nucleus, common services, etc) effected by the fault.

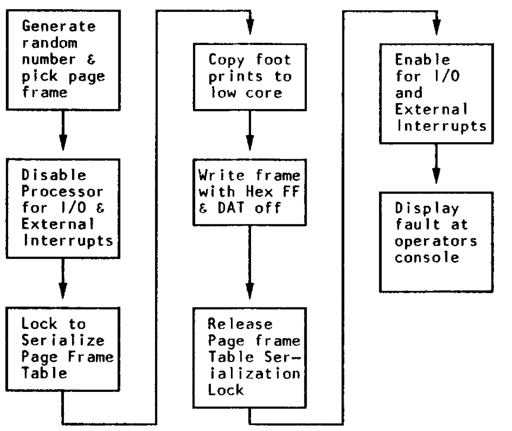
Generate random number & pick page frame
Disable Processor for I/O & External Interrupts
Lock to Serialize Page Frame Table
Copy foot prints to low core
Write frame with Hex FF & DAT off
Release Page frame Table Serialization Lock
Enable for I/O and External Interrupts
Display fault at operators console

Figure 2. Sequence of events to inject a fault.

Between the time the page frame table is referenced and the time we inject the fault, the ownership of the page cannot change. Two precautions are taken. Firstly, the processor running the fault injection program is disabled for both I/O and external interrupts. This is done so the program does not suffer a context switch just after examining the page frame table. Secondly, access to the page frame table is serialized, by obtaining a real storage manager lock, so the frame is not stolen.

The events required to inject the fault are summarized in Figure 2. The fault injection program is run as a started task and waits for the operator to initiate the injection. Once initiated, the program selects a random number between 0-8191 (for 32MB of 4K pages), disables the processor for interrupts, and serializes the page frame table. The selected page frame table entry is copied to a fixed storage location in low core, leaving foot prints, that can be examined with the processor controller in case on an immediate total failure. The fault is injected with DAT off and the serialization lock on the page frame table released. The processor is then enabled for interrupts and a description of the error written to the operator's console.

System, Workload and Experiment Runs

This experiment is conducted using an IBM 3081-KX with 32 Mega-Bytes of main storage. Figure 3 shows the elements of configuration and some performance numbers that illustrate the utilization. The operating system used is MVS/XA version 2.1.3 [IBM84]. The primary application is transaction processing using the Information Management System (IMS) version 1.3 [IBM86]. The disks used by the experiment consisted of ten model 3350 volumes for the MVS and IMS systems. The paging subsystem used eight model 3380 volumes and the IMS databases were contained on 32, model

3380 volumes. The workload had been carefully constructed and calibrated to characterize various attributes of a real data center. The workload is driven by simulated terminals submitting transactions. The system has been tuned to match the transaction rate expected in a real data center for the given workload, software levels, and hardware configuration. In our configuration approximately 80 terminals are used to to bring the processor to a CPU utilization between 85% and 90%. Each experiment was conducted using exactly the same techniques of bringing the system to full utilization.

The experiment is started with a power-on-reset followed by an Initial Program Load (IPL) of MVS and bringing up of the IMS data-base subsystem. The setup process enables us to perform each experiment under identical workload conditions. The process of initializing the workload can take more than an hour if the data-bases need to be restored from tapes. Terminals are added and the workload increased until the the CPU reaches between 85% to 90% utilization. Adequate I/O capacity had been provided to reach this level of utilization. After the system utilization has been steady for some time the fault injection is performed. After fault injection, the machine is monitored by automated data gathering mechanisms in addition to human observation. The system activity display provides an immediate view of the machine activity. This proved important since there could be partial failure of the system identified by symptoms such as decreased CPU utilization. Terminals were also connected to verify availability of services. A maximum of 15 minutes after the fault was provided for observation of the system. From the initial experiments it was clear that we were getting the type of acceleration intended. The failures effecting the system or service did so with very short latencies, of the order of a few seconds or less. The class of errors which did not adversely affect the system for at least 15 minutes would be latent for much longer without the acceleration. The choice of fifteen minutes provided adequate time for a recovery process to take effect.

A total of 70 experimental runs were performed. Of these, 7 runs were discarded due to either imperfect workload conditions or imperfect recording of environmental data. The 63 runs provided a sample size adequate to avoid many small sample problems. Environmental data such as performance monitors, error logs and dumps were collected for each run. Specifically, we used Resource Management Facility Monitor I and II reports, the MVS error log (*logrec*), and the system message console (*syslog*). During the experiment we also used the System Activity Display (SAD), a service provided by the processor controller to display activity of the processors, their supervisor and problem state usage, and activity of the channels. The fault injection was initiated from the MVS console and information on the fault written to the console.

III. ANALYSIS OF FAILURES

We first present the failures resulting from the faults, and then discuss the effects of the failures. This is viewed as it affects the primary service, namely transaction processing. However, not all errors cause failure and we analyze the ones which did not cause a loss of primary service as well as those that were very latent. These are analyzed by tracing the errors back to the source code or owner of the data. This leads to an interesting set of errors, called potential hazards, which are discussed.

```
System                              Utilization/Performance

CPU      : IBM 3081 Model KX        CPU                 85-90%
Storage  : 32 Mega Bytes            Real Storage        90%
Channels: 24                        Interrupts          575/sec
DASD     :                          VIO Paging          580/sec
  10 3350 Vols   MVS and IMS         (4% of interrupts)
  32 3380 Vols   IMS DBase          IMS Trans           26/sec
   8 3380 Vols   Paging             Response Time       0.5 sec
Terminals:                          Address Spaces      55
  Real and simulated                Multiprogramming 12-15

Software                            Workload

OS:    MVS Ver   2.1.3              An online IMS database
Subsystems:                         application that has
  IMS Version 1.3                   been tuned to simulate
  VTAM, Communication               a real data center.
  TSO, Timesharing
```

Figure 3. Experimental Setup.

Failure Classification.

Each failure has been recorded with a variety of attributes and parameters describing both quantitative and qualitatively the effect on the system. This set of failure descriptions have been collapsed to a few failure groups which best identify the significant failure characteristics of the system.

Figure 4 contains a table of the classifications with corresponding percentages for each failure group. *MVS Crash* was when the operating system crashed and an IPL was necessary to restore the system. *IMS Crash* indicates that IMS was lost completely, although MVS was still able to perform dumps and restore the system. The *Crash* cases contribute to a total loss of service. A hot standby would be the most likely recovery strategy to deal with such failure cases. It is interesting to note that only 16% of the total faults caused the entire system to crash. *A single transaction failure* indicates that a failure was contained within a message processing region and IMS fenced it from the rest of the system. The *IMS impaired* was only one instance but it has been maintained as a separate group to illustrate events, when a reduced transaction processing rate was sustained. The *MVS Impaired IMS OK* case includes situations when some significant MVS service was un-available, but the loss did not affect the functioning of IMS. In a data center, such a situation would necessitate an IPL to restore MVS services. An example of such a failure would be the loss of dump services. An extreme example of such a case includes the loss of the master console. Interestingly, the IMS console is separate, so IMS can run unhindered but the system will need an IPL. *Non Service related, IMS OK* includes failures that do not affect IMS or any significant MVS service that IMS requires. An example would be the loss of TSO. *Nothing Happened*, as the name suggests, is a case where there was no observable damage to any subsystem or component. This is an interesting case analyzed in detail later in this section.

FAILURE GROUP	FREQUENCY	PERCENT
1. MVS Crash	2	3.2
2. IMS Crash	8	12.7
3. Single Tran. Failure	8	12.7
4. IMS Impaired	1	1.6
5. MVS Impaired, IMS OK	9	14.3
6. Non Service Related, IMS OK	3	4.7
7. Nothing Happened	32	50.8
	63	100.0

Figure 4. Failure Classification.

Effect on the Primary Service

The impact of a fault in the system can be assessed in many ways. One of the important measures in the area of commercial transaction processing is the availability of a specific service. Research studies have traditionally measured the number of failures in a system, but failures in a large systems do not necessarily affect either the service rendered or the integrity of data. The *loss of service* experienced by a user is relevant in the commercial computing environment. A failure in the system will cause a loss of availability only if the specific service is not available to the user when requested. From the service standpoint, a failure masked by the system or unrelated to the service is of no consequence to the availability of the system. Thus, if in consequence to a failure, the system is reconfigured or the request handled by an alternate mechanism it does not cause an outage, perceived by the user, provided the response time is maintained.

In this experiment the primary service of transaction processing is rendered by the IMS application. MVS has other applications running under it that provide a secondary service, for example, Time Sharing Option (TSO). However, MVS does provides a number of services that are essential to both the primary and secondary applications. To assess the effect of a fault from the service point of view, we consider *no loss of service* to be a case when the primary service is 100% available for a reasonable time after the fault. A damage to a secondary service which does not cause disruption of the primary service is not considered loss of primary service. However, in the long run, damage to the secondary service may cause disruption of the primary service. A *total loss of service* is when IMS is not able to provide the transaction processing service, although MVS may be available for other applications such as TSO. Two cases of a *partial loss* of service are identified. One, when only a fractional part of the total system throughput is available for transaction processing. The other, when a *partial loss* is recovered completely with almost imperceptible loss in performance.

Figure 5 shows a histogram of the four cases, total loss of service, partial loss of service, partial loss of service (which recovered) and no loss of service. Examples of cases with total loss of service are and IMS crash, MVS crash or IMS impaired to an extent wherein the transaction rate is not maintainable. Partial loss of service is only one case, but it has been maintained as a separate case to illustrate the instance of fractional throughput in transaction processing. The cases of partial loss which have been totally recovered are single

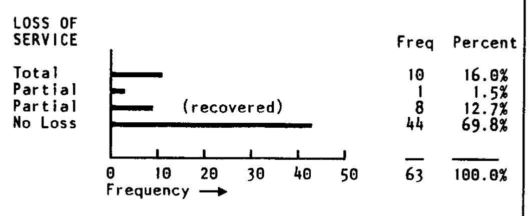

Figure 5. Loss of Service to the Primary Application.

transaction failure occurring in a Message Processing Region (MPR). These failures are trapped by IMS and it fences the failure from the rest of the system by containing it within the MPR. IMS will then restart the MPR to fully restore the system. Under *no loss of service*, the user of transaction processing is not affected and will not have any knowledge of the fault.

A total loss of service resulted in 16% of the faults. Given that the system has been almost maximally accelerated this represents an empirical upper bound to such failures. The acceleration also ensures that faults effecting the system do so with minimum latency. Faults effecting any component, typically did so within 1 second of the fault. Thus, the probability of IMS transaction processing being available in the near term after a fault is at least 0.69. This is interesting since it constitutes a large fraction of the incident faults. Clearly the faults which did not affect the system for up to 15 minutes are a category which need further analysis. In the literature they are commonly referred to as having a large latency and nothing else was known about them. In this paper we analyze these latent errors using data gathered during the experiment.

Figure 6 gives a breakdown of the 44 faults (69.8%) which did not cause any loss of service. *Impaired system, IMS Okay* contains faults affecting the MVS operating system, however, IMS remained unaffected. This group includes faults such as the loss of dump services, and even the master console. In either case the primary service was unhindered. However, in a data center these would eventually require an IPL to restore the non primary service lost. A *non service related damage* would be, for example the loss of TSO. A large number of them fall into the class, *nothing happened*, summarizing the perception of an operator or any client of IMS. These are the subject of discussion in the next subsection.

Analyzing Faults with Very Large Latency

The 32 faults under the group *nothing happened* have been analyzed individually. In each of these cases, a dump of the address space faulted was taken before turning off the experiment. The dump together with the virtual address of the fault give us adequate information to trace the fault. By using logic manuals, source code, debugging manuals etc., we tried to identify the function which the fault effected and evaluate the impact of the fault. It is an elaborate process and in some cases, such as certain private virtual area faults, it was not always possible to evaluate the impact.

'NO LOSS OF SERVICE' CLASSIFICATION	FREQ	PERCENT
Impaired System, IMS Okay	9	14.3%
Non Service Related Damage	3	4.7%
Nothing Happened	32	50.8%
	44	69.8%

Figure 6. Case with *No Loss of Service.*

'NOTHING HAPPENED' CLASSIFICATION	FREQ	PERCENT
Potential Hazard, Identified	14	22.2%
Not Known	12	19.0%
Overwritten (Error → Good) (transitions)	6	9.5%
	32	50.8%

(A)

SOURCE OF POTENTIAL HAZARDS TYPE/STORAGE AREA	FREQ	PERCENT
Local System Queue Area	10	15.9%
Dormant MVS Code	3	4.7%
Other	1	1.6%
	14	22.2%

(B)

Figure 7. The *Nothing Happened* and *Potential Hazards*

This analysis revealed some interesting characteristics of errors. We found a number of faults caused errors contributing to serious damage. At the same time we noticed, given the current working state of the machine, it was very unlikely the error would cause a failure. This interesting combination was a common element shared by large set of errors and has led to our definition of a *potential hazard.* We define it and then discuss its attributes.

Definition - Potential Hazard:
A potential hazard is defined as an error which has caused significant damage, however, under the current operating state the error will remain dormant.

Note that the system has definitely been damaged, however, the potential hazard is one where if the *current operating state* is not changed a failure will not occur. This is an important point, since it opens up a variety of possibilities to deal with the situation. By a change in operating state we mean substantial change in workload or a change in configuration. In our experiment, the potential hazards remained and did not cause failures since the workload was steady. A simple example of a potential hazard is a corrupted device configuration table. An attempt to bring a string of disks on-line can trigger a total failure. The table in Figure 7 (a) shows 22.2% of the faults are identified as potential hazards. The *not known* (19%) are faults which we either could not analyze due to insufficient information, or it did not share characteristics of a potential hazard. However, if completely analyzed, some of them may fit the description of the potential hazard. In 9.5% of the cases, the fault was overwritten returning the machine to a good state. Given failure acceleration it represents a minimum. The table in Figure 7 (b) shows the sources of the potential hazards that were identified. A large fraction of them are from the Local System Queue Area.

It is conjectured that *Potential Hazards* offer an interesting explanation to a phenomenon observed regarding workload and failure dependency. It has been noticed the failure rate in some large installations increase rapidly with corresponding workload increases, specifically between batch and interactive workloads shifts [Iyer82]. Potential hazards trigger failures only when there is a substantial shift in the workload and would contribute to the type of phenomenon noticed.

IV. FAILURE PREVENTION AND ERROR REPAIR

This section brings together a summary of the measurements to identify possible directions which can be followed to deal with failures in large systems. Importantly, we identify a class of errors using relatively inexpensive methods to either perform repair and alleviate an otherwise imminent failure. Additionally, we identify the types of storage areas in which such error groups are likely to exist. These estimates are useful as the basis for design techniques to deal with such errors.

An important observation is that only a small number of faults cause a catastrophic resulting in total loss of the primary service. Figure 8 illustrates the state transitions and the corresponding percentages. For simplicity all percentages quoted are fractions of the total 63 faults. In this section we draw attention to the *partial failures* and *nothing happened* cases. Partial failures occur in 33% of the faults, not all of them affecting the primary service. In fact, 19% are partial failures not effecting the short term availability. Almost half of the faults, 51%, are classified as *nothing happened.* About 10% of these errors are returned to the good state by the system overwriting the error. The remaining 41% are split into *potential hazards* and a group undetermined. The dotted line through the *partial failures, potential hazards* and the *not determined* constitute a total of 60% which share some unique characteristics. These errors have in common that:
- The primary service is not affected.
- There is adequate time for repair.

In addition, it may involve either a *non-service related* failure (19% of all faults) or a *potential hazard.* Under current practice, most partial failures would imply an IPL, thereby leading a total outage. If there has been no failure, but an error exists (though unknown) it could be a potential hazard. Potential hazards imply adequate time, although, a total failure is likely following a change in workload.

The common characteristics of such errors, i.e., the primary service not affected and there exists adequate time, (typically in the order of tens of minutes) provides an opportunity to avoid a total outage. This leads us to define a general recommendation for such errors.

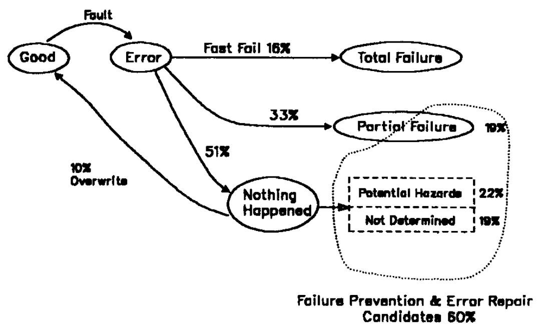

Figure 8. State Transitions under Failure Acceleration.

Definition - Failure Prevention and Error Repair:
Failure Prevention and Error Repair (FPER) is the technique of detecting errors and taking action such that the error can be removed to avert the loss of a primary service.

It is clear that if effective *FPER* mechanisms were designed, there can be a substantial improvement in the availability and MTBF of large systems. Given time is not constrained FPER techniques need not be expensive.

An immediate question is, *what are the kinds of techniques that are applicable* and *where are the FPER candidates?* The answer to the first is very dependent on the second. This section addresses the second question, i.e., *where are the likely FPER candidates?* We mention, in passing, that FPER techniques can include code refreshing, partial IPL, subcomponent restart, resilient data structures etc.

FPER Storage Area Distribution.

One way to look for FPER candidates is to know the likely virtual storage areas locations. The storage area which the fault affects influences the type of repair action that can be initiated. We have collapsed storage areas into three groups to best illustrate their characteristic and present them in Figure 9. The columns of the matrix identify three groups pf storage areas and the rows the FPER possibilities. The 63 faults have been assigned to one of the 9 bins. Note FPER is only applicable when there is not a fast fail or a loss of the primary service. The the first row are non FPER candidates. The second rows contain the errors where FPER is not possible and the third row where FPER is possible. To achieve this classification, each error has been analyzed through a sequence of classifications and only summarized in this table. What may appear as an FPER *candidates* in the earlier figure, may not be classified as one that is FPER *possible* in this table.

The virtual storage areas have been collapsed to three groups. Nucleus (NUC) and the Link Pack Area (LPA) are together since most of it is code. The Common Service Area (CSA) and System Queue Area (SQA) are together since they are predominantly data. The third group is the private areas of address spaces. It can be seen that private area faults are the bulk (53%) of the FPER possible, but note that it is only a third of all the private area hits. The NUC and LPA are the second largest group that are FPER candidates, yet are more than two thirds of all errors in that group.

Code versus Data Damage.

A common question arising is the effect of faults on code versus data. Code errors lend themselves to different fault tolerance techniques as compared to data errors. For instance, code errors can be repaired by code refreshing since most code is reentrant. Whereas, errors in data need more elaborate techniques to identify and correct them. We separated the errors into two groups namely code and data to look at some possible differences.

The distribution of the loss of service (such as Figure 5) was similar and almost identical for errors due to faults in code or data. This indicates that the relative chances of a total, partial, or no loss of services is independent of whether the fault occurs on code or data. The potential hazards were classified as due to either code errors or data errors to reveal the respective conditional probabilities. We find, that given a potential hazard, the conditional probability that data has been damaged is much higher than code. Specifically at a 95% confidence level:

$$P(\mathit{Code\ Damage}|\mathit{Potential\ Hazard}) = 0.2 \pm .15$$
$$P(\mathit{Data\ Damage}|\mathit{Potential\ Hazard}) = 0.8 \pm .15$$

This provides direction for developing techniques to avert potential hazards. Resilient data structures and other methods to identify control block failures will significantly reduce potential hazards.

In summary, there exist are a large fraction of errors which are candidates for *failure prevention and error repair*. Techniques to implement FPER will significantly boost availability of the primary service.

KEY: Frequency, % of All, Row %, Col %	VIRTUAL STORAGE AREA			
	CSA+SQA Area	NUC+LPA Area	Private Area	Row TOTALS
Failed - FPER Not Applicable	1	2	7	10
	1.59	3.17	11.11	15.87
	10.00	20.00	70.00	
	16.67	15.38	15.91	
FPER Not Possible	3	1	23	27
	4.76	1.59	36.51	42.86
	11.11	3.70	85.19	
	50.00	7.69	52.27	
FPER Possible	2	10	14	26
	3.17	15.87	22.22	41.27
	7.69	38.46	53.85	
	33.33	76.92	31.82	
Column TOTALS	6	13	44	63

Figure 9. FPER Virtual Storage Area Distribution.

V. CONCLUSIONS

This paper provides a unique insight into the failure of large systems.

- We have introduced the technique of *failure acceleration* which provides a systematic method to design fault injection experiments. Maximum failure acceleration estimates empirical bounds for state transition probabilities of the failure process.
- The fault injection experiment conducted on a commercial transaction processing system found that a total loss of the primary service (as a fast fail) occurred in only 16% of the faults. Given failure acceleration, this estimates the maximum that may occur in practice.
- Analysis of errors revealed a class of errors termed *Potential Hazards* numbering at least 22%. Potential hazards are guaranteed not to affect the short term availability and will cause a catastrophic failure only when there is a significant change in the operating state of the system. It is conjectured that these can contribute to the cause of workload dependent failures reported in literature.
- At least 41% of errors are identified as potential candidates for what we call *failure prevention and error repair*. These errors share two necessary conditions. One, they either do not affect the short term availability or only cause a non-service related damage. Two, there is adequate time (in the order of tens of minutes) for repair action. Currently most such errors result in an IPL of the system, causing a total outage. However, their characteristics provide an opportunity for inexpensive techniques to recover from such errors. Estimates are provided for the likely location of such errors in terms of storage area and code-data split.

Acknowledgments

George Wang spent a lot of time discussing critical aspects of this work and projecting the impact of results. Phil Rosenfeld has been a strong source of encouragement. We also thank the anonymous referee who provided the thorough review.

References

[Chillarege87] R. Chillarege and R. K. Iyer.
Measurement based Analysis of Error Latency.
IEEE Transactions on Computers, 36(5):529-537, May 1987.

[Endres75] A. Endres.
An Analysis of Errors and Their Causes in System Programs.
IEEE Transactions on Software Engineering, 1(2):140-149, 1975.

[Hummel88] R. A. Hummel.
Automated Fault Injection for Digital Systems.
Proc. Annual Reliability and Maintainability Symposium, 1988.

[IBM84] IBM.
MVS/Extended Architecture Overview.
1984.
(GC28-1348).

[IBM86] IBM.
IMS/VS Version 2 General Information Manual.
1986.
(GC26-4180).

[Iyer82] R. K. Iyer, S. E. Butner, and E. J. McCluskey.
A Statistical Failure/Load Relationship: Results of a Multi-Computer.
IEEE Transactions on Computers, 31(7):697-706, July 1982.

[Mourad87] S. Mourad and D. Andrews.
On the Reliability of the IBM MVS/XA Operating System.
IEEE Tran. Software Engineering, 13(10):1135-1139, Oct 1987.

[Segall88] Z. Segall, D. Vrsalovic, and et.al.
FIAT - Fault Injection Based Automated Testing Environment.
Digest, Eighteenth International Symposium on Fault Tolerant Computing, pages 102-107, 1988.

View publication stats