Fault Manifestation Model for Predicting Anomalous System Behavior

Lu Luo
School of Computer Science
Carnegie Mellon University
luluo@cs.cmu.edu

Abstract

While there are an enormous number of faults that can occur in the hardware and software of a computing system, several research studies have indicated that the number of fault manifestations is usually very small, often resulting in less than a dozen different states. The paper surveys studies of both naturally occurring and artificially induced faults. A composite set of faulty states is proposed as well as transition probabilities between states. The transitions with the largest range of valued are identified. Future work will simulate the model and do a sensitivity analysis on it to determine the critical transitions. The model will be used to study the placement of fault detection monitors that will predict failure in a computer system.

1. Introduction

The pervasive employment of computer systems in critical applications over the past decades has made fault tolerance a vital issue. To design fault tolerant systems, it is essential to know the sources and frequency of errors, how faults manifest into errors, and what is the relative probability of each manifestation. However, faulty behaviors of complex computer systems, especially in distributed real-time applications, have not been well characterized or understood.

In this paper we present a model of hardware/software fault manifestations at the system level. A standalone computer system is modeled as a Finite State Machine (FSM), where the states are a set of normal and faulty situations of the system, and the transitions, labeled with conditional probabilities, are a set of the events that cause the state transitions. The studies on which the fault model is based cover various platforms including different versions of Windows and the most widely used implementations of POSIX operating systems [1], indicating the model's potential for generality across platforms. Our model shows that there are orders of magnitude fewer manifestations than actual faults.

Based on this model, the future faulty behavior of a real computer system can be subsequently monitored and predicted by an external monitor. Therefore, appropriate reconfiguration behavior could be initiated prior to catastrophic computer failure. The model will also be extended to include network behavior so that a complete multi-computer distributed network system can be modeled.

2. Identification of Fault Manifestations

The FSM model was constructed by surveying previous fault injection studies. Four studies of fault manifestations formed the basis for the model [1,2,4,5]. The fault manifestations are aligned in Table 1, where similar categories between the studies are horizontally aligned.

Table 1. Classification of fault manifestations

Cristian '91	Barton '90	Suh '92	Koopman '00
Omission failures	No Error	Result returned (No error detected)	OK/Normal
Timing ▪ Early ▪ Late	Response too late	Late response – Timeout	–
Response ▪ Value ▪ State	Invalid output	Failure – Incorrect answer	–
Crash ▪ Total amnesia	Machine crash	Crash	Crash
▪ Partial amnesia ▪ Pause-crash ▪ Halt-crash	Task stop (Process crash)	Abort (Crash with error message)	Abort
–	–	–	Reboot
–	–	–	Silence
–	–	–	Hindering

Cristian introduced a theory of the basic classification of fault manifestations in distributed system architectures [5]. Though Cristian's classification was developed for distributed systems, there is a strong mapping to single user systems. In [2], Barton et. al. presented the experimental results of automatic fault injection. Three separate types of memory bit faults were injected at various locations in the target system. The results

showed that there are a limited number of system-level fault manifestations. Suh et. al.[4] were concerned with creating a benchmark to measure system robustness rather than a theoretical approach to defining fault manifestations, as was the focus of Cristian's work. In the work of Koopman and colleagues [1], Ballista automatically generates test to measure the robustness of COTS operating systems. The anomalous response of the observed systems had been classified into a "C.R.A.S.H." scale, as is shown in Table 1. Each of these studies covered different types of machines and system topologies, as well as portions of a system hardware/software component. The notable overlaps in the mapping of the error states in these studies indicate the possibility of modeling a computing system with a generic composite set of states with properly chosen granularity.

Besides the experimental results from artificially induced faults, several studies of naturally occurring faults also served to provide the transition probabilities between the states. In [8], the Andrew File System servers of CMU were observed for 21 workstation years while [7] served over 300 machine years of data.

3. The Fault Manifestation Model

The fault manifestation model (FMM) is a 4-tuple finite state machine $M = (S, T, s, s_0)$, where S is the set of states; T is the set of events that cause transitions between states; $s: S \times T \rightarrow S$ is the transition function; and s_0 is the initial state. The fault injection studies in Table 1 were used to define the states in the FMM depicted in Table 2.

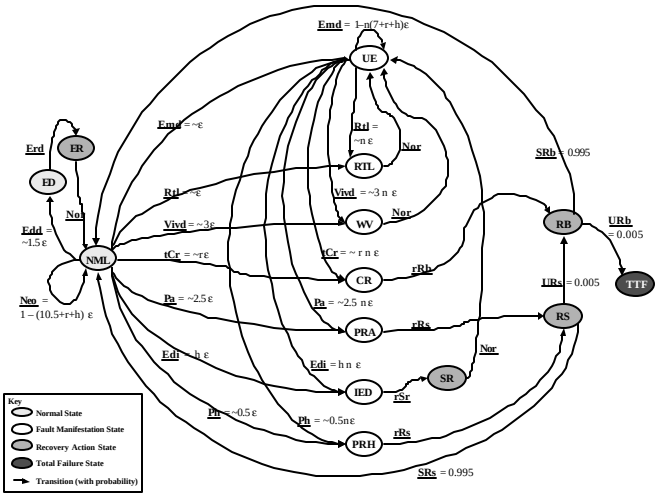

Figure 1. Fault Manifestation Model

The Fault Manifestation Model (FMM) is depicted in Figure 1. Four types of states are included in the FMM: normal, fault manifestation, recovery action, and total failure. The normal states are NML and ED, which represent the regular activities a system performs, including system error detection. There are seven fault manifestation states that summarize the 25 states identified in the four studies in Table 1: UE, RTL, WV, CR, PRA, IED, and PRH. There are four error recovery action states: ER, SR, RS, and RB. It is worth mentioning that when the system enters TTL, the state of total failure, the normal state cannot be resumed without replacement or repair.

Table 2. List of States in the FMM

State		State Description
Normal	NML	Normal. System operates in normal state.
	ED	Error detected. By either the system error detection mechanisms or the data comparison task.
Fault Manifestation	UE	Undetected Error. System returns on indication of error on an exceptional operation. Latent fault and errors still reside in the system.
	RTL	Respond too late. System does not deliver its data or service within the specified timeout period.
	WV	Wrong value. The output value a system gives is different from what is expected.
	CR	Crash. System stops working and a reboot is need.
	PRA	Process abort. Failures result in abnormal termination.
	IED	Incorrect error detection. The system detected the error but gave out incorrect error code.
	PRH	Process hung. Process never gives any output, nor can it terminate by itself.
Recovery Action	ER	Error recovery. System recovers from detected error.
	SR	Software recovery. System hires software error recovery mechanism to fix the detected error.
	RB	Reboot. System needs to reboot.
	RS	Restart. Process or task needs to be restarted.
	TTF	Total failure. The system cannot resume any operation even after reboot.

A description of the transitions between the states in the FMM is listed in Table 3. The conditional probabilities of error events that cause the state transitions of the FMM are based on the previous studies in Section 2. Since these studies were carried out in different systems with different methods, evaluated ranges of the probability of some transitions are given. It is observed that the average system crash rate is in the range of (0.002, 0.2), the reboot rate is in the range of (0, 0.1), and the process abort rate in the range of (0.2, 0.5). The transitions with the largest range of value are identified by algebraic analysis. All transition arcs are parameterized in terms of one constant e, and three variables r, n, and h.

- Constant e is a small exponential number and e. is defined to be the ratio of error events to normal events in the system. It is estimated that normal events

happen on the order of 10^5 more frequently than error events, so we define $e = 10^{-5} = 0.00001$

- Variable n is larger than 1 and n is defined as the ratio of undetected errors to the ones that are detected. The value of n depends on the error detection rate of specific system. Based on the estimated value we gave to e, the value of n should be larger than $1/e$, which is 10^5
- Variable r is defined as the probability (frequency) that the system will go to crash. A range of (0.02, 0.2) is suggested for r
- The variable h is defined as the probability that the system goes into hindering state, an incorrect error code is returned, and the transition Edi is hired. It is suggested that h be in the range of (0.1, 1)

The rest of the transitions are defined to have fixed probabilities. As the system must enter an error recovery state after each fault manifestation states, the probability of corresponding transitions is 1. For better visibility, the probabilities of these transitions are not marked in the graph. From Table 3 we can see that transitions Erd, Nor, rRb, rRs, and rSr have a probability of 1. The "~" in some transition probabilities represents approximation.

Table 3. List of Transitions

Transition	Description	Probability
Neo	Normal system event	1-(10.5+r+h) ε
Edd	Error detected	~1.5 ε
Emd	Error missed detection / Silence	~ε, 1-n(7+r+h) ε
Rtl	Respond too late	~ε, ~nε
Vivd	Invalid value or output	~3ε, ~3nε
tCr	To crash	~ rε, ~rnε
Pa	Process aborted	~2.5ε, ~2.5 nε
Edi	Error detected incorrectly / Hindering	hε, hnε
Ph	Process to hung	~0.5ε, ~0.5nε
Erd	Error recovered	1
Nor	Normal operating resumed	1
rRb	Requesting reboot	1
rRs	Requesting restart	1
rSr	Requesting software recovery	1
SRb	Successful reboot	0.995
SRs	Successful restart	0.995
URb	Unsuccessful reboot	0.005
Urs	Unsuccessful restart	0.005

4. Conclusions and Future Work

Faults can occur any place in a computer system and at any time. The cross product of space and time is computationally prohibitive. The number of fault manifestations is much smaller. In this paper we developed a model based on fault manifestations that was derived from previous natural and artificial fault monitoring experiments. The model identifies the error states of a generic computer system, the transitions between the system states, and the conditional probabilities of each transition. The model is definable by only four variables: e, n, r, and h.

At current stage, the model presented in this paper is still very preliminary. Several steps remain. First, a simulation evaluation of the model over the ranges for the variables is needed to determine which variables are most sensitive. Second, apply the after the model to monitor the system fault states in real time, and to predict future error behaviors. The model can also be used to direct the reconfiguration of system. Third, extend the model to include the fault manifestations of distributed systems with multiple nodes.

5. References

[1] P. Koopman and J. DeVale, "The Exception Handling Effectiveness of POSIX Operating Systems", *Transactions on Software Engineering*, IEEE, Vol. 26, No. 9, September 2000, pp. 837 - 848.

[2] J. Barton, E. Czeck, Z, Segall and D. Siewiorek, "Fault Injection Experiments Using FIAT," *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990, pp. 575 – 582.

[3] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownackj, J. Barton, R. Dancey, A. Robinson, and T. Lin, "FIAT – Fault Injection Based Automated Testing Environment", *Eighteenth International Symposium on Fault-Tolerant Computing*, 1988, pp. 102 –107.

[4] B. Suh, J. Hudak, D. Siewiorek, and Z. Segall, "Development of a Benchmark to Measure System Robustness: Experiences and Lessons Learned", *Third International Symposium on Software Reliability Engineering*, 1992, pp. 237 - 245.

[5] F. Cristian, "Understanding fault-tolerant distributed systems", *Communications of the ACM*, vol. 34, No. 2, February 1991, pp. 56-78

[6] D. Siewiorek and R. Swarz, *Reliable Computer Systems: Design and Evaluation, Third Edition*, A K Peters, Ltd. Natick, MA, 1998.

[7] M. Buckley, "Computer Event Monitoring and Analysis", *PhD Thesis*, Carnegie Mellon University, May 1992.

[8] T. Lin and D. Siewiorek, "Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis", *Transactions on Reliability*, IEEE, Vol. 39, No. 4, October 1990, pp. 419 - 431.