Fault Tolerance and Reliability Techniques

Lecture #15: Tuesday, May 20 2003
Lecturer: Arjun Singh and Ernesto Staroswiecki
Scribe: Honggo Wijaya and Navneet Aron

1. Introduction

Fault tolerance issues are becoming more relevant in systems today due to technological and architectural reasons. As wires become denser and feature sizes get smaller, systems are more susceptible to transient errors due to crosstalk and bit-flips due to alpha particles. Moreover, as the number of components increase and designs become very aggressive, systems are more fault prone as the verification of these designs may not be exhaustive.

Hence, we need architectural solutions to improve availability. Traditionally, availability is provided by forward error Recovery which adds redundant hardware to mask faults at a high cost or by Backward Error Recovery which rolls back to a consistent state on detecting a fault (at the expense of performance).

Today, we will look at 2 proposals to address this issue: SafetyNet and Fault-tolerant Design of the IBM pSeries 690 System.

2. Paper 1 : SafetyNet

SafetyNet is an availability solution which uses a unified checkpoint/recovery mechanism to support multiple long-latency fault detection schemes. They claim to provide all three goals of high availability, high performance and low cost – something which traditional fault tolerant systems cannot guarantee.

SafetyNet only targets transient faults like dropped messages or semi hard faults like half a network switch dies and all its buffered messages are lost. They cannot handle permanent faults like chip kill or the full switch dying or even uncorrectable faults where the Error Correcting Code (ECC) architectural state is affected.

SafetyNet's goal is to recover to a consistent state when a fault is discovered with minimum performance degradation. Barring performance degradation, this can be achieved naively by periodically quiescing the entire system by taking checkpoint; while

taking a checkpoint, including the entire state of the machine, and stopping the system to validate the checkpoint as fault-free. SafetyNet optimizes the above three steps to minimize overhead. It efficiently coordinates a periodic creation of checkpoints across the system using a "logical time base" with the help of service controllers. In order to minimize amount of storage, it only logs the first store/transfer per block per checkpoint interval. Finally, it does a pipelined validation of checkpoints while the system is still running.

Implementation: They simulated a 16 node, 4 by 4 torus with the following parameters on the following applications:

- Simulation
  - Simics full-system simulation of 16-proc SPARC system
  - Detailed timing simulation of memory system

- MOSI directory cache coherence protocol
  - Simple, in-order processor model
  - 128KB L1I/D, 4MB L2, 512KB CLB

- Workloads (commercial and scientific)
  - Online transaction processing (OLTP): IBM's DB2
  - Static web server: Apache driven by SURGE
  - Dynamic web server: Slashcode
  - Java server: SpecJBB
  - Scientific: barnes-hut from SPLASH2

Results show that SafetyNet gives very little performance degradation for the common case when there is no fault and only up to 10% degradation in the case when a hard fault or 10 transient faults occur per second for a 1 GHz processor.

A summary of its strengths and weaknesses:

Strengths:

- SafetyNet: global, consistent checkpointing
  – Low cost and high performance
  – Efficient logical time checkpoint coordination
  – Optimized checkpointing of state
  – Pipelined, in-background checkpoint validation
  – Avoid crash in case of fault
  – Same fault-free performance
  – Can tolerate long detection latency
  – No s/w modification required
  – Low o/p commit latency

Weaknesses:

- Only tolerate some permanent faults
  – Not chip kills, CLB bit flips or faults that cause ECC architecture state corruption.

- Processor modification required
  – Add CN field to cache lines
  – Add CLB buffers.
- Assume a separable NS and EW switch
  – What happens if its not separable – I.e. the whole switch dies.
- Do not talk about a generic formula for
  – CLB size = f (check pt interval length, program event generation)
  – 512 KB only good enough for the 5 applications they try at 100Khz checkpoint frequency.

3. Paper 2: Fault-tolerant design of the IBM pSeries 690 system using POWER4 processor technology

3.1 Overall description

This paper describes, mostly, the hardware techniques used in the IBM pSeries 690 system for fault detection, isolation, and debugging. It also briefly explains some recovery techniques, as well as some of the needed software interaction to achieve the goal of fault-tolerance.

3.2 System description

The IBM pSeries 690 is a general-purpose UNIX server, based on IBM's flavor of UNIX called AIX. It consists of up to 4 boards, with 4 POWER4 chips per board. Each POWER4 chip has 2 processing cores (8-way multi-scalar), private L1 caches, a shared L2 cache, and L3 directories to interface with an on-chip L3 cache. All this creates a 32-ways system.
The fault-tolerance design goals are to provide high levels of reliability, availability, and serviceability. In order to achieve this, their system needs to be able to detect failures, recover from failures, and isolate failures. The key fault-tolerance design characteristics of this system allow it to detect errors during normal machine operation, to capture the machine state when an error occurred, and to isolate the source of failure. Also, through error checkers, they are able to provide data integrity, initiate the recovery mechanisms when an error has been detected, and deterministically isolate physical faults.

3.3 Fault-tolerance techniques

3.3.1 Field Replaceable Unit isolation

In other to provide good serviceability it is desirable to isolate a fault to a single Field Replaceable Unit (FRU). To maximize the times this happens the IBM system includes error checkers immediately before and after any FRU-FRU interface, and therefore only if an error is not detected before the interface but is afterwards, we have some ambiguity (the error can be on the driver, the interface, or the receiver).

3.3.2 Active Source Identifier

This is simply a latch that stores the source of data to a particular register that has an error checker. With this, we can directly identify the error as coming from one source, and not from all the many that could be arriving to this register.

3.3.3 Who's On First

Through deterministic analysis the system's firmware can determine which error checker came on first (in case more than one came on) and therefore determine the original source of the fault. This limits the logical error domain of a checker, and allows for finer diagnosis through the hierarchical examination of the Fault-Isolation Registers (FIRs).

3.3.4 Service Processor

This is a separate, independent processor, that is transparent to the systems, and has the following tasks: to receive attention signals from the hardware, to read and write to the FIRs (to diagnose a problem, or to make sure that the correct data is there for posterior analysis), to do a first diagnosis of a problem (implementing the WOF method), to write the error state to a NVRAM, and, finally, to call the OS or the firmware to trigger recovery.

3.3.5 Memory Fault-tolerance

The memory system contains ECCs (Hamming, detect two, correct one). Every error found is treated as an interrupt, recoverable or not. When an error is detected, an error counter is incremented. If this counter goes over a threshold, we considered this memory to be defective and it is disabled.
To deal with permanent errors, the system includes spare bits, as well as steering logic to be able to use them. Also, the bits of a word are spread out over different chips, and therefore the system can even survive memory chip-kill.

3.3.6 System Recovery

This is triggered when loading data with an error. If the error is recoverable, the system simply invalidates the corrupt data and tries again. If it is not, the state is saved, and the OS is called in to determine the nature of the affected process and terminate it. Through Logic Partitioning (LPAR) only the affected partition is terminated. One interesting detail is that when data recovery still does not give the correct results, the data is tagged as defective, but only when this data is referenced is that we get a system error.

3.3.7 PCI Bus/Devices

The system now provides mechanisms for PCI Bus fault-tolerance, like instruction level re-try, or single bit fault detection. However, these techniques need to be exploited by the device drivers when they interact with their devices, and therefore are out of the scope of IBM.
Similarly, AIX provides the device drivers with the ability to write to the error log, and act the same way itself does when detecting an error in their devices. For both these reasons, IBM encourages fault-tolerant behavior from its device drivers providers, but cannot enforce it.

3.3.8 Minimum Impact Availability

The system is capable of deconfiguring almost any part (with some granularity) considered to be defective, either at run-time or boot-time. This allow for the system to keep on working and not being bogged down by repairing errors constantly in the presence of several permanent errors.

3.4 Strengths

IBM is a company known for its fault-tolerant design, although mostly in their mainframe products.
Therefore, when they apply many of their proven techniques for a High-end UNIX server, you need to believe that the end result will be an extremely reliable machine.
The fault-detection techniques described in this paper are extremely thorough, providing excellent detection and isolation capabilities. This ability, together with the saving of the state whenever an error occurs, allow for a very detailed debugging process, without the need of replicating the error.
Their diagnosis capabilities are good enough that they can almost always isolate a fault to a single Field Replaceable Unit, and one of the famous phrases attributed to IBM service personnel is “Hi, I am here to replace the board that failed last week and you did not notice...”

3.5 Weaknesses

The weaknesses described here are not about the techniques themselves, but about the paper. In brief, the paper has almost no numbers, and the very few that are there seem almost thrown in, with no justification.
The only number in the papers refers to the 95% of faults that are isolated to a single Field Replaceable Unit. There is no analysis of performance in the presence of different error rates, how many and what types of error the system can stand without crashing, etc., etc.
It would be really nice to see numbers indicating, for example, the mean time between killer failures, the hardware, power, or performance overhead due to the fault detection mechanisms, etc., but I guess this is what you get when you read a paper from industry. (Note that this paper is from the IBM journal of research, I don’t think any refereed publication would have accepted such a paper without any experimental results.)

4. Discussion

4.1 SafetyNet and IBM pSeries fault tolerance mechanism

What kind of errors do we target in SafetyNet?
SafetyNet only provide recovery mechanism (decoupled from the detection mechanism) and targets transient faults like dropped messages or semi hard faults like half a network switch dies. It cannot handle permanent faults like chip kill or the full switch dying

What does Safety Net do on errors?

In the occurrence of errors, SafetyNet basically rolls back to a state which is Most Recently Validated Checkpoint which is also called Recovery point. Then, it resumes the execution from that point. The advantage of such a scheme is low overhead.

How is this scheme different from IBM pSeries 690?
IBM pSeries 690 provides Redundancy. It has more hardware. It employs a voting mechanism to detect error; therefore this scheme has higher overheads.

What are various ways of detection of errors?
ECC, CRC, timeouts, repetition with voting to check results, creating of exceptions for unusual events. For example, a scheme could suspect bit flipping and trigger an error.

What are various types of errors?
- Soft error
- Hard error
- Heisenburg -> error that rarely happens and can't be reproduced easily
  - Ex : Race conditions

What type of errors are attacked in SafetyNet and IBM pSeries?
The IBM pSeries 690 is able to deal with all types of errors mentioned above.
Heisenburg errors are easy to be catched with the IBM pSeries method since all the error states will be logged to be analyzed later on.

SafetyNet only targets a limited type of errors as mentioned above.

The other main difference between SafetyNet and IBM pSeries is: IBM pSeries also count the error. Thus, if a certain error on a specific component happens too frequently, that particular component can be disabled while SafetyNet only log the error and then roll back to the previous recovery point. Thus if an error happening over and over, SafetyNet won't be able to recover it.

4.2 Key to Reliable System

The most important feature of a reliable systems is that there should be no single point of failure. For example in case of IBM pSeries 690, the processor is the single point of failure.

IBM pSeries 690 design approach is an overall System design idea. The concept is that every component should work together.

In designing such systems, making models of expected errors can help because then the system design does not need to take care of all unexpected errors. For example memory correction of 20 errors is not needed because its occurrence is almost non existent.

Also if the recovery period after the fault takes a significant amount of time, there should be mechanism to tolerate faults during the recovery period.

Stress testing is also very important so that any faulty chips will collapse before it is shipped to the customer. Thus, only the most reliable components get shipped.

Both papers have a commonality in the fact that both do error recovery in the background in order to avoid interrupting the user processes.

This raises the question whether we can and should expose some of this fault diagnosis/recovery to the Operating System.
Both the papers mention that they do make the OS aware of that. In some cases, it may be exposed to the application files system. For example, transaction processing/ database applications have their own mechanism of recovering from error.

The next question being raised is what capabilities the applications should have to help the fault tolerance mechanism.
Any application which provides fault tolerance should have:

- Atomic Transactions
- It should be able to repeat the computation on triggering of error recovery mechanism
- It is also a good idea to have hardware support for such purposes(faster)

How does Safety Net handle fault tolerance on I/O?
SafetyNet handles fault tolerance on I/O by allowing the I/O to only see states that are already validated. Therefore, having too many unvalidated states will lead to longer latency.

What are the special considerations in doing Fault Tolerance in the context of CMP?

Advantages:

- Multiple processor core and memory provide inherent redundancy
- The modularity provides some isolation mechanism. The signals communication about failures can move fast, for example if there is a central unit which is responsible for collecting all information regarding failure.

Disadvantages:

- More complex hardware, more sources of errors.
- Harder to probe a specific node because many processors are packed in a single chip. -> solve this using built-in logic analyzer in the chip.

Different hardware designs of CMP aid differently in fault tolerance/isolation.
Finer granularity is usually better. For example in TRIPS, if one of the core processor has an error, we need to disable ¼ of the chip, while in Smart Memories disabling one core only disabling 1/16 of the chip.
Another issue is about recompilation. Statically scheduled code (such as in RAW) needs to be recompiled before it can be executed after a faulty component is isolated.

High power consumption (ex: TRIPS) also leads to higher probability of transient errors. In case of errors due to leakage, higher capacitance could be used but the downside is it increases the power consumption.

The case of fault tolerance in memory is a little bit different than in the core processor. Memory is relatively cheap, therefore we can provide lots of redundancy. The structure is also very regular (consists of rows and columns). We can provide double columns and rows to provide redundancy.

References

1. D. Bossen, A. Kitamorn, K. Reick, M. Floyd. Fault-tolerant Design of the IBM pSeries 690 System using Power4 Processor Technology. IBM Journal of Research and Technology, Volume: 46, Issue: 1, January 2002.

2. D. Sorin, M. Martin, M. Hill, D. Wood. SafetyNet; Improving Availability of Shared Memory Multiprocessors with Global Checkpoint/Recovery. Proceedings of the 29th International Symposium on Computer Architecture, Anchorage, AL, May 2002.