FAULT TOLERANT SYSTEMS

http://www.ecs.umass.edu/ece/koren/FaultTolerantSystems

Part 5 – Processor-Level Techniques & Byzantine Failures

Chapter 2 – Hardware Fault Tolerance

Copyright 2007 Koren & Krishna, Morgan-Kaufman

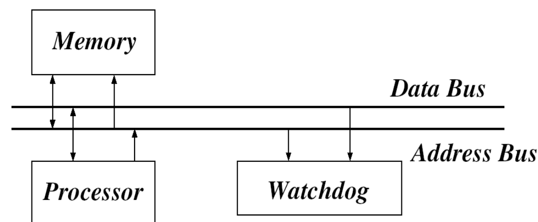
Processor-Level Techniques

- Duplicating processors is often prohibitive
- Simpler techniques with smaller overheads exist:
- 1. Reexecuting the program – time redundancy
  - Up to 50% performance loss
- 2. Instruction Retry
  - Requires fault detection circuitry
  - Effective for transient faults
  - Low performance overhead – transparent to program
- 3. Monitoring by a simpler "watchdog" processor
- 4. Simultaneous Multithreading for Fault Tolerance

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Watchdog Processor

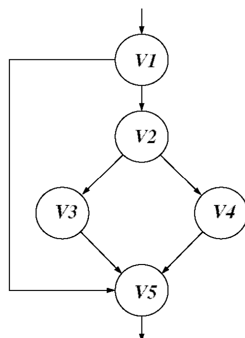
- Performs concurrent system-level error detection
  - Monitoring bus connecting main processor and memory
  - Targets control-flow checking: correct program blocks executed in the right order
  - Can detect hardware/software faults causing erroneous instructions to be executed or wrong execution paths
  - Watchdog needs the program's control-flow information



Copyright 2007 Koren & Krishna, Morgan-Kaufman

Watchdog – Control-Flow Information

- Derived from Control-Flow Graph (CFG) of program
- Assign signatures to CFG nodes & store in watchdog
- Run-time signatures computed & compared to stored
  - Signatures assigned (e.g., successive integers) or calculated
  - Assigned can detect invalid execution paths (e.g., {V1,V4})
  - Can not detect errors in instructions



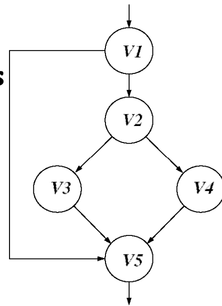
Assigned signatures:

```
accept sig(V1);
either
    accept sig(V2);
    either
        accept sig(V3);
    or
        accept sig(V4);
    accept sig(V5);
or
    accept sig(V5);
```


Copyright 2007 Koren & Krishna, Morgan-Kaufman

Calculated Signatures

- ♦ Calculate signature from instructions in block, e.g., add them (mod 2) or checksum code
- ♦ Calculating run-time signatures & comparing to stored ones can detect erroneous instructions
- ♦ Most data errors will not be detected
- ♦ Improvement: include "assertions" (acceptance tests) to be checked by watchdog
  - * Need forwarding of variables to watchdog and ability to verify assertions
  - * Alternative: use parity to check data
  - * Current superscalers are difficult to monitor – accessibility and speculative fecthing

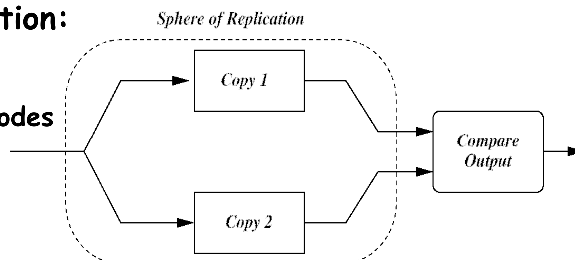


accept & check sig(V1);
either
  accept & check sig(V2);
  either
    accept & check sig(V3);
  or
    accept & check sig(V4);
  accept & check sig(V5);
or
  accept & check sig(V5);

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Simultaneous Multithreading for Fault Tolerance

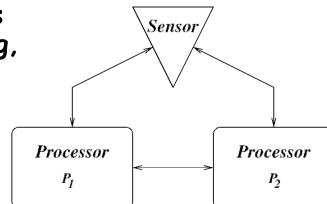
- ♦ Create two independent & identical threads for each application thread
  - * Comparing their results will detect transient faults
  - * To reduce (performance) penalty 2nd thread trails the 1st
  - * Pass information from leading thread to trailing, e.g., results of conditional branches
- ♦ To support independent execution some hardware duplication is required, e.g., architecture registers
- ♦ Sphere of replication:
  - * Units not included in sphere can use error-detection codes



Copyright 2007 Koren & Krishna, Morgan-Kaufman

Byzantine Failures

- Devices can exhibit malicious faults and produce arbitrary outputs – Byzantine faults
  - No problem in M-of-N systems - voter masks out erroneous outputs
  - Distributed system without a centralizing entity may have problems
- Example: A sensor providing temperature to two processors, using point-to-point links
  - Sensor has suffered a Byzantine failure and tells processor P1 that the temperature is 25 degrees and P2 that it is 45
  - P1 and P2 can exchange messages and know that something is wrong, but not who is faulty – P1, P2, or the sensor



Copyright 2007 Koren & Krishna, Morgan-Kaufman

Byzantine Generals (or Interactive Consistency) Problem – General Model

- A single entity (such as a sensor or processor) sends the value of some variable to a set of receivers
- The receivers can communicate among themselves to exchange information about the value they received
- A functional unit is truthful in all its messages
- A faulty unit may send out contradictory messages
- Time-out mechanism detects absence of a message
- We are looking for an algorithm which will satisfy
  - IC1: All non-faulty units must arrive at an agreement of the value that was transmitted by the original source
  - IC2: If the original source is non-faulty, the value they agree on must be the value that was sent out by the original source

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Byzantine Generals (or Interactive Consistency) Algorithm

- Original (and simplest) algorithm is recursive
- N units (one source and N-1 receivers); Up to m faulty
- Interactive consistency obtained only if $N \geq 3m+1$
- The algorithm Byz(N,m) has three steps:
- Step 1:
  - Original source sends information to each of N-1 receivers
- Step 2:
  - If $m>0$, each of the N-1 receivers runs Byz(N-1,m-1) to send the value it received to the other N-2 receivers
  - If a unit does not get a message from another unit (after some time-out), it enters the default value into its records
  - If $m=0$ this step is bypassed
- Step 3:
  - Each receiver now has a vector of m values and uses the majority, or the default if no majority exists
  - If $m=0$, it uses value received from original source

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Interactive Consistency Algorithm - Notation

- A and B are units - A.B(n) means A sent B the message n
- U is a string of units $A_1, A_2, \ldots, A_m$ - U.B(n) means B received n from A_m who claims to have received it from A_{m-1} and so on
- φ - A message that is not sent, e.g., A.B(φ) means the message that A was supposed to send B was never sent
- A.B.C(n) - B told C that the value it received from A was n
- A.B.C.D(n) – D received n from C who claims to have received it from B who, in turn, claims to have received it from A
- Black.White.Green(341) means Green received the message 341 from White who claims to have received it from Black

- Example: m=0 - Degenerate case: no fault tolerance
  - Step 2 bypassed – Interactive Consistency Vector (ICV) consists of a single value that has been received from original source

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Example: m=1

- Must have at least 3m+1=4 units: sensor S + 3 receivers, R1,R2,R3
- Sensor is faulty
  - sends out inconsistent messages to receivers: S.R1(1), S.R2(1), S.R3(0).
  - All receivers are functional
  - default is 1
- Step 2: R1,R2,R3 each acts as source and runs Byz(3,0)
  - following messages are sent:
  - S.R1.R2(1) S.R1.R3(1)
  - S.R2.R1(1) S.R2.R3(1)
  - S.R3.R1(0) S.R3.R2(0)
- Element j of ICV at receiver Ri is equal to
  - Report of Rj as determined by Ri if $i \neq j$
  - Value received from original source if $i=j$
- At the end of this step, ICVs are (1,1,0) at every receiver
- Majority vote yields 1 - the value used by each of them

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Example: N=7, m=2 (R1 and R6 faulty)

- Messages sent in first round by S are consistent
  - S.R1(1), S.R2(1), S.R3(1), S.R4(1), S.R5(1), S.R6(1)
- Each receiver executes Byz(6,1) in step 2 of Byz(7,2)
- R1 is faulty and can send out any message (or nothing) - it sends in step 1 of Byz(6,1):
  - S.R1.R2(1) S.R1.R3(2) S.R1.R4(3) S.R1.R5(4) S.R1.R6(0)
- In step 2 of Byz(6,1) each of R2-R6 uses Byz(5,0) to disseminate the message received from R1 - the messages are:
  - S.R1.R2.R3(1) S.R1.R2.R4(1) S.R1.R2.R5(1) S.R1.R2.R6(1)
  - S.R1.R3.R2(2) S.R1.R3.R4(2) S.R1.R3.R5(2) S.R1.R3.R6(2)
  - S.R1.R4.R2(3) S.R1.R4.R3(3) S.R1.R4.R5(3) S.R1.R4.R6(3)
  - S.R1.R5.R2(4) S.R1.R5.R3(4) S.R1.R5.R4(4) S.R1.R5.R6(4)
  - S.R1.R6.R2(1) S.R1.R6.R3(8) S.R1.R6.R4(0) S.R1.R6.R5(φ)
- R6 being maliciously faulty is free to send out anything it likes

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Example – Cont'd (R1 and R6 faulty)

- ICVs at each receiver in connection with the S.R1(1) are:
  - $ICV_{S.R1}$(R2) = (1,2,3,4,1)
  - $ICV_{S.R1}$(R3) = (1,2,3,4,8)
  - $ICV_{S.R1}$(R4) = (1,2,3,4,0)
  - $ICV_{S.R1}$(R5) = (1,2,3,4,0)
  - $ICV_{S.R1}$(R6) is irrelevant, since R6 is faulty
- R5 received nothing from R6, its value is recorded as the default, say 0
- R2, R3, R4, R5 find no majority in ICV and assume the default value (0) for S.R1
- Similarly, agreement can be reached on the message that S sent to each of the other receivers
- This completes the generation of the ICVs connected with the original Byz(7,2) algorithm

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Message Authentication

- The Byzantine Generals problem is hard because faulty processors could lie about messages received
- Simplify by providing a way to authenticate messages
  - Each processor can append to its messages (included forwarded ones) an unforgeable signature
  - The recipient can check the authenticity of signatures
- A message forwarded through A and B, can be checked to see whether the appended signatures of A and B are valid
- All processors have a time out mechanism against faulty processors that remain silent

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Algorithm with Message Authentication

- Algorithm ABYz(N,m) (for maintaining interactive consistency):
- Step A1: Original source signs its message ψ and sends out to each processor
- Step A2: Each proc. i that receives a signed message $\psi:A$ (A - set of signatures appended to ψ), checks # of signatures in A
- If $\# < m+1$ - sends out $\psi:A \cup \{i\}$ (adds its own signature) to each processor not in A & adds ψ to list of received messages
- Step A3: After seeing signatures of all processors (or time-out), apply some decision to select from messages received

Copyright 2007 Koren & Krishna, Morgan-Kaufman

Outline of Proof for N>m+2

- Case 1: Original source functional - a signed message μ transmitted to every processor
- Nobody can forge the source's signature, no processor will accept message other than μ in Step A2
- Case 2: Original source faulty - different messages may be sent out to different processors, each with a correct signature
- List of received messages (minus signatures) is the same at each processor
- Proof by contradiction

Copyright 2007 Koren & Krishna, Morgan-Kaufman