Fault Tolerance

SafetyNet:
Improving the Availability of Shared Memory Multiprocessors with Global Checkpoint/Recovery

Fault-tolerant design of the IBM pSeries 690 system using POWER4 processor technology

-Arjun Singh
-Ernesto Staroswiecki

Outline

- Availability
  - Motivation
  - Example targeted faults
  - Difference between SafetyNet and other systems
- SafetyNet: Key Features
- A SafetyNet Implementation
- Evaluation
- Conclusions

Availability Motivation

- Fault frequencies are increasing

1. Technological reasons
   - Smaller transistors
   - Denser wires
2. Architectural reasons
   - More components
   - More aggressive designs

- Marketing trends demand more availability

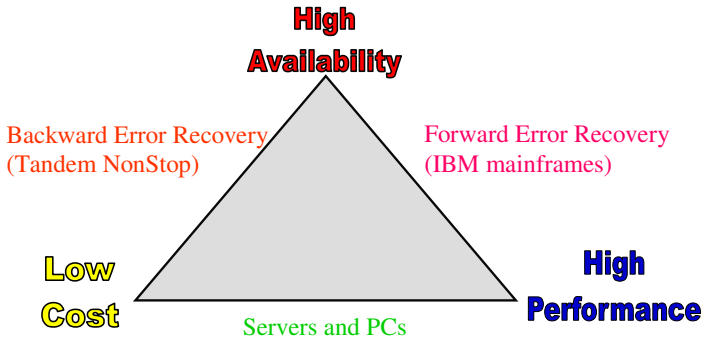
Need architectural solution to improve availability

Traditional Availability

- Forward Error Recovery (FER)
  - Use redundant hardware to mask faults
  - E.g., triple modular redundancy with voter or pair&spare
  - Systems: IBM Mainframes, Intel 432, Stratus
  - Sacrifices cost to achieve availability
- Backward Error Recovery (BER)
  - If fault detected, recover system to pre-fault state
  - Periodically stop system and save state or log changes
  - Fault? Restore pre-fault checkpoint or unroll log
  - Systems: Sequoia, Synapse N+1, Tandem NonStop
  - Sacrifices performance to achieve availability

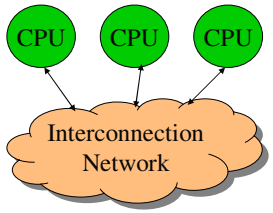
System Hardware Design Space

Existing systems get only 2 out of 3 features



Which Faults Does SafetyNet Target?

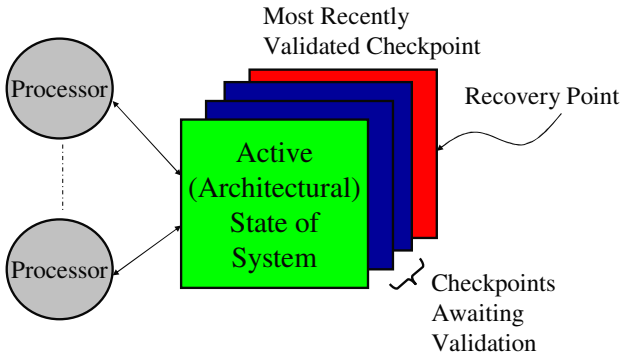
- Hardware faults in shared memory multiprocessors
  - Mostly transient, some permanent, not chipkill
- Focus on faults outside of processor cores
  - Why? Good techniques for processors (e.g., DIVA)
- Interconnection network
  - Example: dead switch
  - Detect with timeout
- Cache coherence protocols
  - Example: lost coherence message
  - Detect with timeout



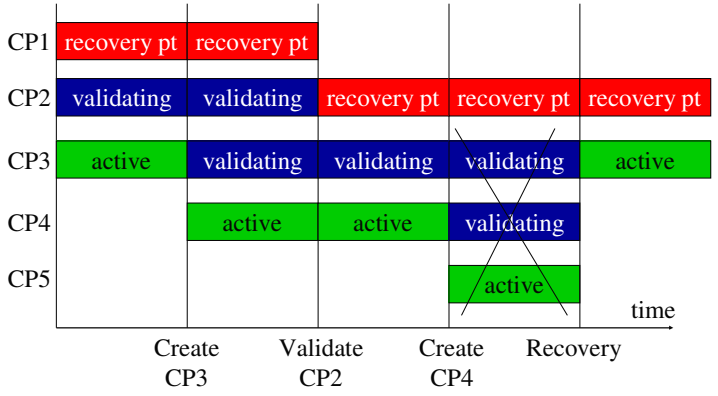
Outline

- Availability
- SafetyNet: Key Features
  - System abstraction
  - Innovations
- A SafetyNet Implementation
- Evaluation
- Conclusions

SafetyNet Abstraction



SafetyNet Execution Model



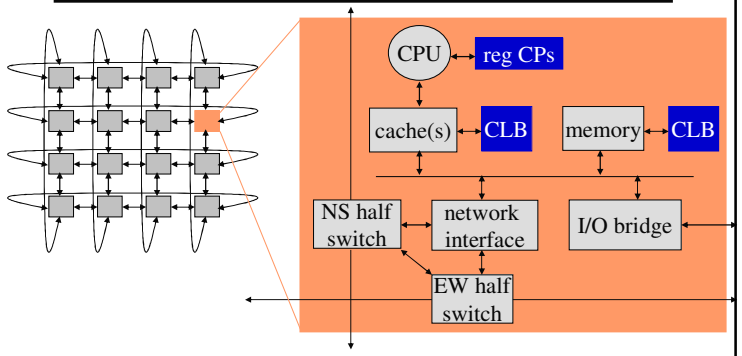
SafetyNet Goal and Innovations

- Goal: Recover to consistent checkpoint if fault
- Inefficient but correct solution
  - Periodically quiesce entire system to take checkpoint
  - Checkpoints include all system state
  - Stop system to validate checkpoints as fault free
- SafetyNet innovations:
  1. Efficient coordination of checkpoint creation across system
     - Use logical time base for checkpoint creation
  2. Optimized checkpointing of system state
     - Only log first store/tranfer per block per interval
  3. Pipelined validation of checkpoints in background
     - Can hide long fault detection latencies - Number of outstanding checkpoints x checkpoint length
     - Design tolerance to be longer than longest detection latency

Outline

- Availability
- SafetyNet: Key Features
- A SafetyNet Implementation
- Evaluation
- Conclusions

System Model



- Checkpoint Log Buffer (CLB) at cache and memory
  - Just FIFO log of block writes/transfers

System Recovery and Restart

- Any component can trigger recovery
  - E.g., processor times out on coherence request
- All in-progress transactions are dropped
  - By definition, these transactions are not validated
- After recovery, resume execution
  - May have to reconfigure (e.g., route around dead link)
  - Must replay work that was lost

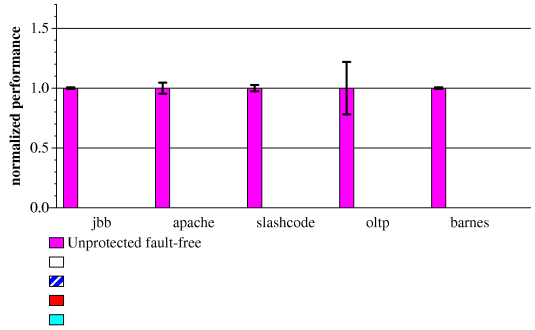
Outline

- Availability
- SafetyNet: Key Features
- A SafetyNet Implementation
- Evaluation
  - Methodology
  - Runtime performance
- Conclusions

Methodology: Simulation & Workloads

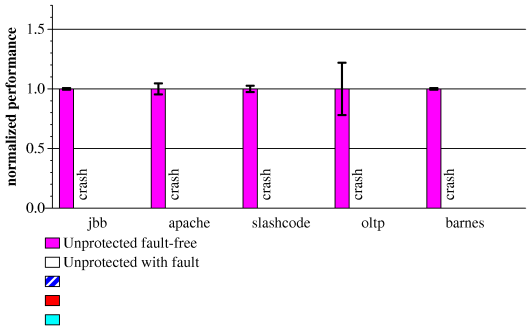
- Simulation
  - Simics full-system simulation of 16-proc SPARC system
  - Detailed timing simulation of memory system
    - MOSI directory cache coherence protocol
  - Simple, in-order processor model
  - 128KB L1I/D, 4MB L2, 512KB CLB
- Workloads (commercial and scientific)
  - Online transaction processing (OLTP): IBM's DB2
  - Static web server: Apache driven by SURGE
  - Dynamic web server: Slashcode
  - Java server: SpecJBB
  - Scientific: barnes-hut from SPLASH2

Runtime Performance



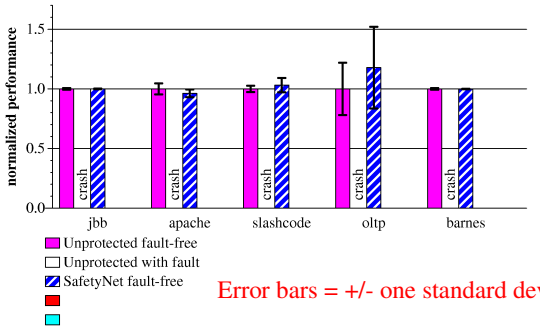
Normalize results to unprotected system

Runtime Performance



Unprotected system crashes if fault occurs

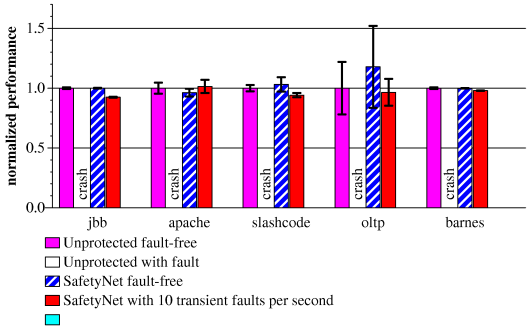
Runtime Performance



Error bars = +/- one standard deviation

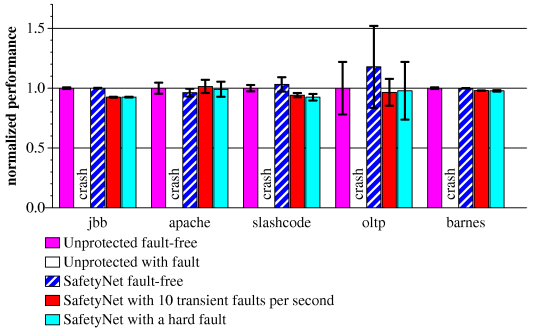
SafetyNet has same fault-free performance as unprotected

Runtime Performance



SafetyNet avoids crashes in presence of lost messages

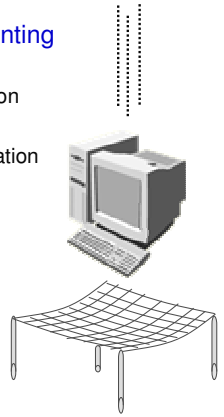
Runtime Performance



SafetyNet avoids crashes in presence of dead half-switch

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.

IBM pSeries 690

- General-purpose UNIX server
- Uses POWER4 chips
  - 2 cores per chip
  - Shared L2 cache
  - L3 directory
  - Up to 4 chips per board
  - Up to 4 boards per node
  - Total of 32 ways

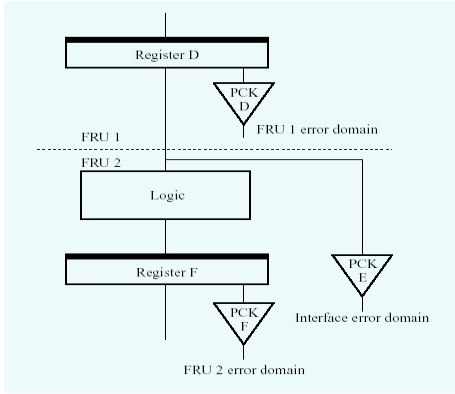
Design goals/characteristics

- Fault-tolerant design to:
  - Detect failures
  - Recover from failures
  - Isolate failures
- High level of:
  - Reliability
  - Availability
  - Serviceability
- Key design characteristics:
  - Detect errors during normal machine operation
  - Capture machine state
  - Isolate source of failure (up to an FRU) by analysis of state
- Error checkers:
  - Provide data integrity
  - Initiate recovery mechanisms
  - Deterministically isolate physical faults

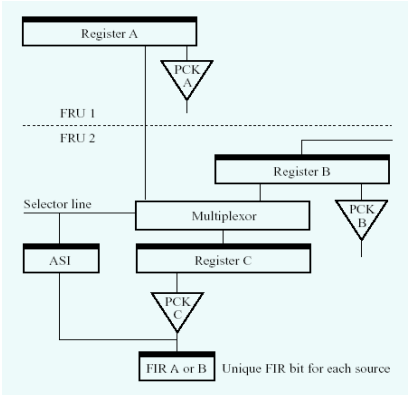
Fault isolation

- Isolate faults to an FRU
  - Allows for quick replacement of failing part
  - Minimize cases where this is not possible
- Strategic placement of error checkers
  - Unique, deterministic, error domain
- FIR
  - Fault-isolation registers
  - Software readable
  - Blocking

FRU Isolation: Signals across boundaries



FRU Isolation: Active Source Identifier

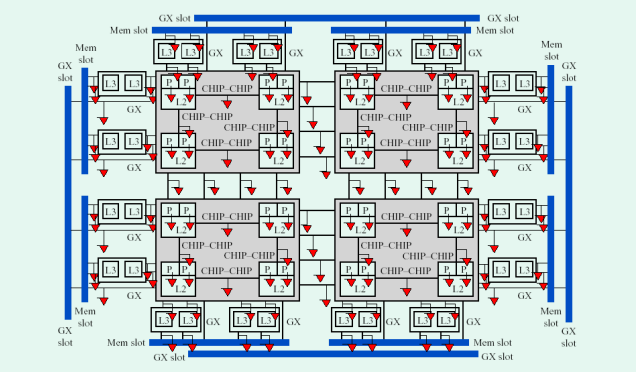


More Isolation

- Who's On First (WOF)
  - Determine which error checker came on first
  - Logically limits error domain of a checker backwards to the previous checker
  - Hierarchical structure of FIR examined sequentially
- Error checker placement
  - System error-checker
  - Isolation internal to the FRU
  - Error information is written permanently on the chip
    - Allows for root cause analysis without re-creating failure

More Isolation

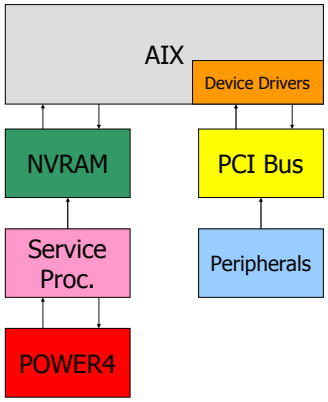
- Sorry, I just love this figure…



Service Processor

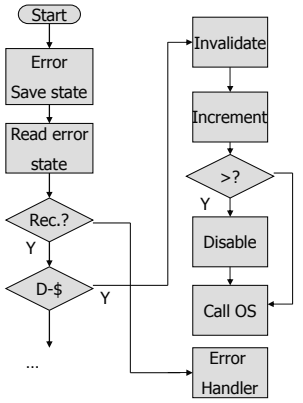
- Separate, independent processor
- Receives attention signals from hardware
- Read/write to the FIRs
- Transparent to system
- Does diagnosis (reading FIRs and using WOF)
- Writes the state in NVRAM
- Calls OS (AIX) and Firmware

Structure of error communications



Memory Fault Tolerance

- SERs are 5000 times greater than intrinsic failure rates
- ECC (Hamming, correct one, detect two)
- Errors are reported as interrupts
- Also deals with permanent errors:
  - Redundancy with spare bits
  - If error count is over a threshold can use steering logic to replace faulty bits
- Main memory fault tolerance
  - Spreads bits across chips
  - Can stand a chip kill



System Recovery

- Triggered by loading data with error
- Synchronous machine-check interrupt
- Recoverable:
  - Described before
- Unrecoverable:
  - Restore processor architected state
  - Call back to the OS
- OS recovery:
  - Determine nature of affected process
  - Terminate process
- LPAR: Logic Partitioning
  - Only affected partition is terminated
- If Data recovery does not give correct results
  - System checkstop
  - Corrupt data is tagged
  - System error handling when corrupt data is referenced

Even more fault tolerance

- PCI Bus:
  - Intermittent errors: Command-level retry
  - Permanent errors: Device driver supported system recovery
  - Fault isolation state is captured
- Minimum impact availability
  - We can deconfigure a failed element
  - With some granularity
  - At run-time or boot-time

Strengths

- Very thorough
- IBM is well known for their fault tolerant systems
  - So we have to believe them it works.
- “Hi, I am here to replace the processor board that will fail in a couple weeks”
- Allows for very good debugging
- Allows for deferred repairs
- Run-time self diagnosis
- Allows a lot of flexibility for SW handling

Weaknesses

- Does not show how to detect processor faults
  - It only shows checks on registers and memories, but does say that processor checks exist.
- What is the overhead?
  - Hardware?
  - Software?
- What are the performance implications?
- What are the fault-tolerance metrics?
  - What kind and how many faults can it stand?
- Any numbers at all?!

Discussion

- What kinds of faults are there/do we want to be able to tolerate?
- What kinds do this two systems tolerate?
- What steps do we need/want in a fault-tolerant system?
- How/what strategy do SafetyNet/IBM use?
- Are there other strategies?

Discussion

- How are the two fault tolerance models in SafetyNet and IBM different?
- What are the assumptions on the faults and fault detection mechanism for SafetyNet to guarantee reliability?

Discussion

- How does SafetyNet coordinate validation of checkpoints?

Discussion

- Can SafetyNet progress deadlock due to full CLBs?

Discussion - I/O and the Outside World

- How does SafetyNet/IBMp690 handle I/O issues?
  - Output commit problem –
  - Input commit problem –

Discussion – CMP fault tolerance

- How would you efficiently use CMP resources for fault tolerance?
- What is easier and what is harder to do in a CMP system?

Backup slides

Key #1
Coordinating Checkpoint Creation

- Checkpoints must reflect consistent system state
  - Nodes must agree on memory values and coherence
- Coordinate checkpoints in logical time
  - Logical time is time base that respects causality
- Each node maintains its own logical clock
  - Create checkpoint every K logical cycles

Need logical time base that helps coordination

Key #2
Optimized Checkpointing of System State

- Insight: only recover at checkpoint granularity
- Intervals between checkpoints group writes/transfers
  - E.g., checkpoint every 100,000 cycles (100 µsec at 1GHz)
- Only log first store/transfer per block per interval
- Optimization at cache:
  - Label cache blocks with checkpoint numbers (CNs)
  - If write/transfer is from same checkpoint, no logging needed

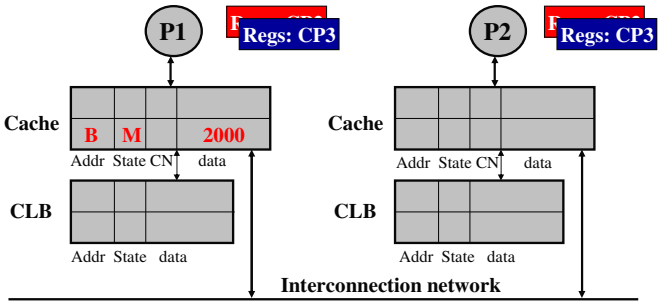
Large benefit due to locality of references

Key #3
Checkpoint Validation in Background

- Only validate when all agree checkpoint is fault-free
  - Example: no outstanding coherence requests in checkpoint
- Nodes perform fault detection, then coordinate
- Can be in background and pipelined
  - Reason why SafetyNet has checkpoints awaiting validation
- Can hide long fault detection latencies
  - Number of outstanding checkpoints x checkpoint length
  - Design tolerance to be longer than longest detection latency

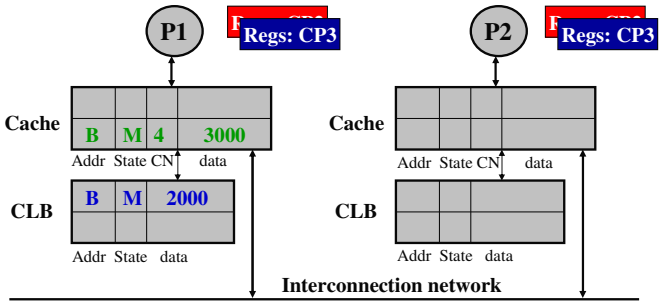
Don't slow down execution to validate checkpoints

Example of SafetyNet Operation



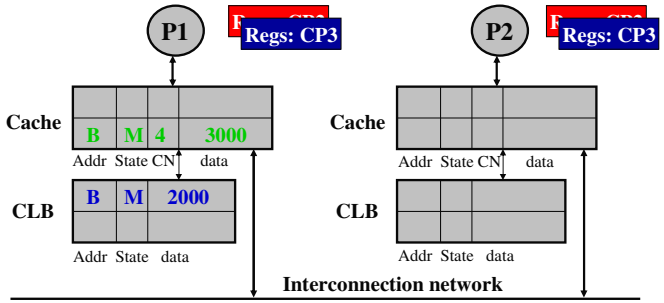
Recovery point is checkpoint 2. Most recent checkpoint is 3.
Active checkpoint is 4. Processor 1 owns block B (validated).

Example of SafetyNet Operation



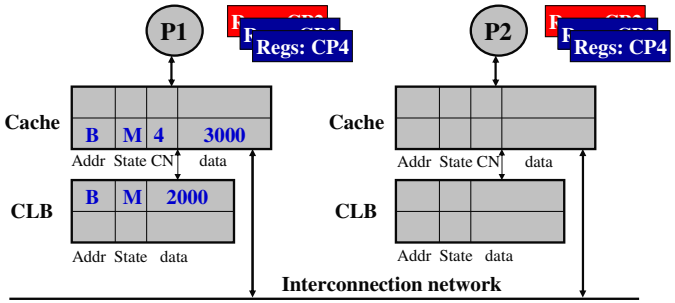
P1 stores 3000 to block B between checkpoints 3 and 4.
Logs old data.

Example of SafetyNet Operation



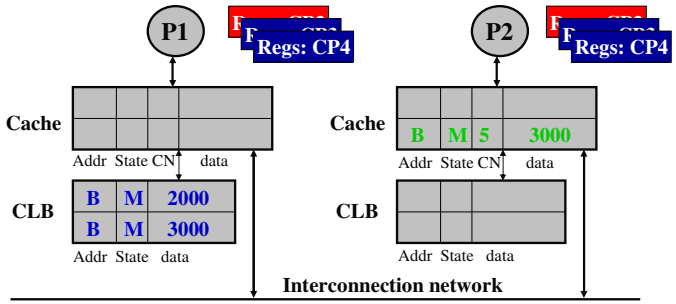
P1 loads from block B.
SafetyNet uninvolved.

Example of SafetyNet Operation



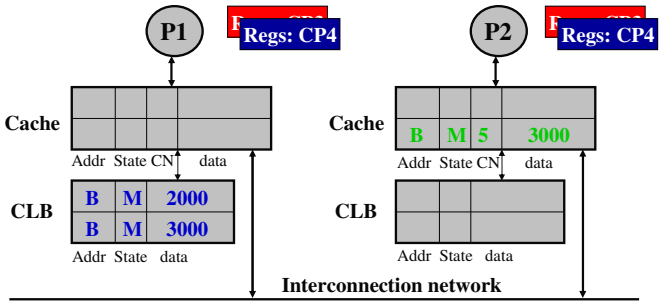
Coordinated creation of checkpoint 4. Active checkpoint is 5.
Save register state at beginning of checkpoint 4.

Example of SafetyNet Operation



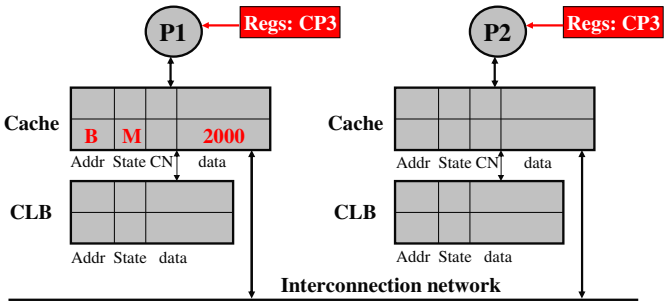
P2 requests ownership of block B. P1 logs old data and sends copy to P2. P1 invalidates cache entry.

Example of SafetyNet Operation



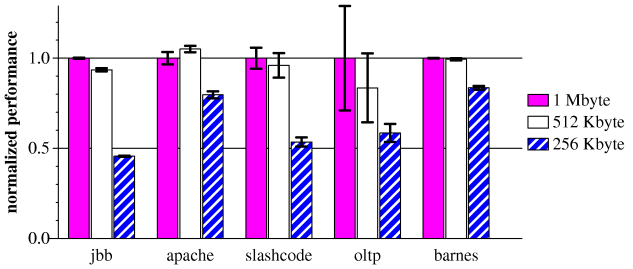
Validation of checkpoint 3. Discard checkpoint 2 registers.
Recovery point is now beginning of checkpoint 3.

Example of SafetyNet Operation



Recovery (to checkpoint 3). Restore CP3 registers. Restore ownership of B to P1. Invalidate B at P2. Now restart system!

Performance vs. CLB Size



Caveat

100,000 cycle intervals

Overview

- Hardware fault frequencies are increasing
- Hardware checkpoint/recovery for multiprocessors
  - Transparent to software
- SafetyNet Innovations
  - Efficient coordination of checkpoint creation
  - Optimized logging of checkpoint state
  - Checkpoint validation off critical path
- SafetyNet achieves 3 goals, existing systems get 2
  - High availability
  - High performance
  - Low cost

Discussion

- How are the two fault tolerance models in SafetyNet and IBM different?
  - SafetyNet uses Backward Error Recovery.
    - But optimizes recovery time.
    - Purely hardware solution.
  - IBM uses hybrid Forward/Backwords Error Recovery.
    - Hybrid h/w s/w solution.
- What are the assumptions on the faults and fault detection mechanism for SafetyNet to guarantee reliability.
  - Faults are not uncorrectable.
  - There is an underlying efficient fault detection mechanism.
  - Fault detection must continue during recovery.

Discussion

- How does SafetyNet coordinate validation of checkpoints?
  - Every component informs the service controllers that its ready to advance its globally consistent recovery point.
  - Service controller broadcasts new recovery check point number (RCPN) to each component and each component updates it.
  - There may be skew in the receipt of this broadcast.
  - The max skew between 2 nodes < min communication latency between them.

Discussion

- Can SafetyNet progress deadlock due to full CLBs?
  - No, The CLB eventually either deallocates state from a checkpoint validation.
  - Or will deallocate due to recovery triggered by service controllers if validation times out.

Discussion - I/O and the Outside World

- How does SafetyNet/IBMp690 handle I/O issues?
  - Output commit problem – Can't send uncommitted data beyond sphere of recoverability
  - SafetyNet includes processors, memory, coherence
  - Doesn't include network, disks, printer, etc.
  - SafetyNet solution: wait to communicate with I/O
  - Only send validated data to outside world
  - Higher pending validations of CNs ➔ higher output commit latency.
  - Input commit problem – Input can't be recovered
  - Standard solution: log input and replay