Lecture 22: Fault Tolerance

Papers:

- Token Coherence: Decoupling Performance and Correctness, ISCA'03, Wisconsin
- A Low Overhead Fault Tolerant Coherence Protocol for CMP Architectures, HPCA'07, Spain
- Error Detection Via Online Checking of Cache Coherence with Token Coherence Signatures, HPCA'07, Duke

Faults

- Faults can be permanent or transient
- Transient faults are typically caused by high-energy particles and noise in voltage levels
- Faults lead to two main errors: silent data corruption, detected unrecoverable error
- A coherence protocol can yield errors in many ways:
  - The delivered message contains corrupted data
  - A message is never delivered
  - The coherence controller computes incorrectly
  - Corrupted state/data in caches
  - Human design error

Token Coherence

- Each memory block has N tokens
- A cache may read the contents of a block if it has at least one token
- A cache may write to a block only if it has all N tokens
- Simplifies the design of a correct protocol: the above abstraction makes it easier to reason about correctness; much easier than reasoning about every corner case as we did for various conventional protocols
- Some inefficiencies: can’t silently evict the block – must send the token to the memory controller first
- Need mechanisms to handle starvation
- One token is flagged as owner and is accompanied by valid data

Fault Tolerant Protocol (Paper #1)

- Assumes an underlying token coherence protocol
- Fault model: only handles faults in the interconnection network; the faults manifest as dropped messages (either the message never arrives or it is discarded because CRC indicates corrupted data)

Potential Problems

- Loss of a token-less message (invalidation) will eventually cause a timeout and the invocation of a persistent request
- Loss of a message with a token will eventually cause a deadlock when the next writer attempts to collect all tokens
- Loss of a message that contains the owner token and data will end up deadlocking and causing loss of valid data

Timeouts

- The token coherence protocol includes a timeout for retry and persistent requests – these will be invoked in case an invalidation message is lost

- If the above fail, another timeout signals a deadlock (potential loss of token) and invokes a token recreation process

- It is possible that a token was never lost, so the token recreation process must be careful to not increase the number of tokens

Token Recreation Process

- The memory controller ensures that the requesting cache ends up with a valid copy of data with all tokens – slow, but correct

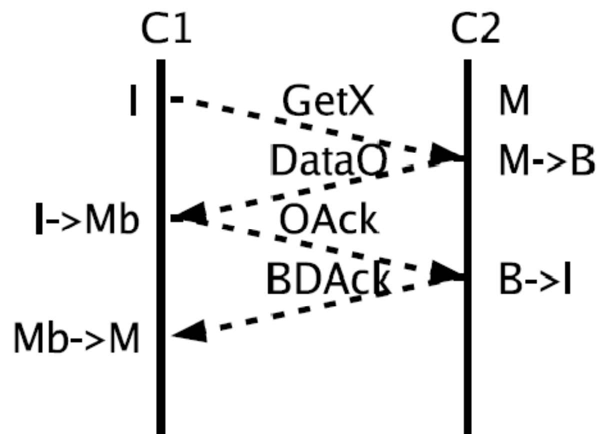
- Since tokens may be stuck in traffic (deadlock false positive), the token-count invariant must not be violated when this token is eventually delivered – hence, tokens have serial numbers and a new serial number is used by the token recreation process

- The memory controller first cancels all existing tokens, informs every node of the new serial number, and collects valid copies of data; then creates the new serial number and tokens, and passes data+tokens to requestor

Backup Data

- If a message carrying ownertoken+data is lost, the only valid copy of data may be lost

- The block is not evicted from the sender's cache until it receives an ack for the above message (the block may be placed in a small backup victim cache)



- The OAck and BDAck need not hold up the write, but they will hold up the transfer of the owner token to the next writer – the system avoids multiple backup copies to simplify recovery

Summary

Fault / Lost message	Effect	Detection and Recovery
Transient read/write request	Harmless	
Response with tokens	Deadlock	Lost token timeout, token recreation
Response with tokens and data	Deadlock	Lost token timeout, token recreation
Response with a dirty owner token and data	Deadlock and data loss	Reliable transfer of owned data using acknowledgements, lost data timeout
Persistent read/write requests	Deadlock	Lost token timeout, token recreation
Persistent request deactivations	Deadlock	Lost persistent deactivation timeout, persistent request ping
Ownership acknowledgement	Deadlock and cannot evict line from cache	Lost data timeout
Backup deletion acknowledgement	Deadlock	Lost backup deletion acknowledgement timeout

Results

- No injected faults in experiments below
- Potential performance loss: less cache space because of backup copies; can't service the next write request until the backup is deleted

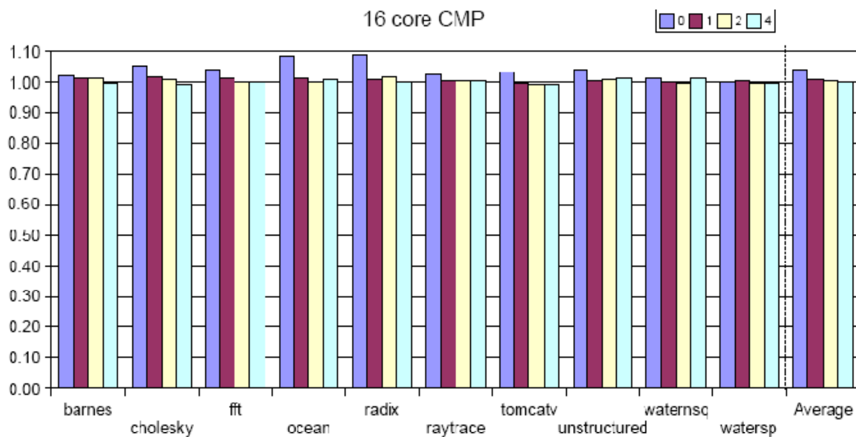


Figure 3. Execution time overhead of our proposal compared to TokenCMP for several backup buffer sizes.

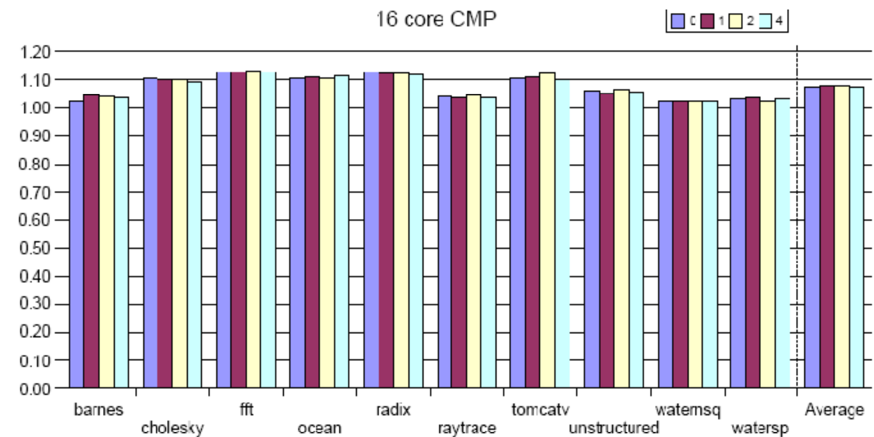


Figure 4. Network traffic overhead of our protocol compared to TokenCMP.

Results with Message Loss

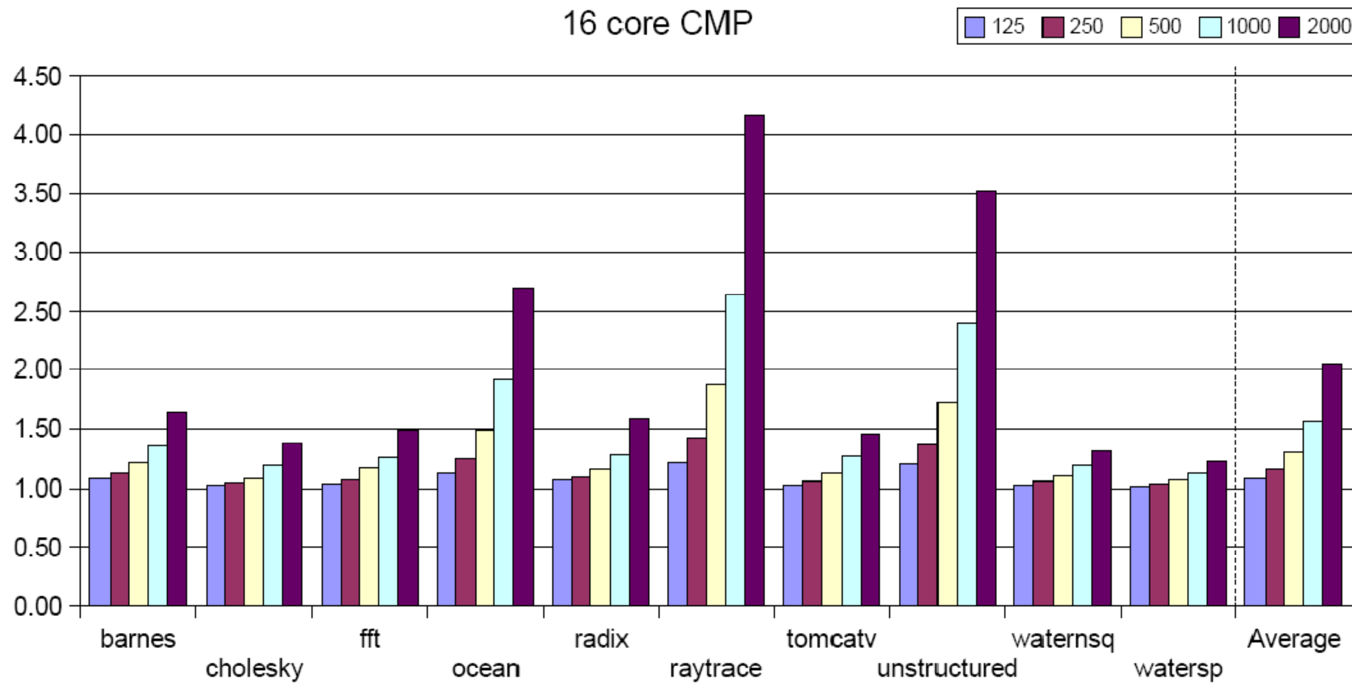


Figure 5. Execution time overhead under several message loss rates.

Token Coherence Signatures

Errors can be detected by checking each invariant in token coherence:

- Each block has T tokens, of which one is the owner token
  - there are initially T tokens for a block in the system
  - a node can never hold <0 and >T tokens for a block
  - if a node sends N_t tokens for a block at time t, another node must receive N_t tokens for that block at time t (transfers are not instantaneous, but we will assume that the receiver owned the tokens since time t)
- A processor can write only if it has all T tokens
- A processor can read a block only if it has at least one token and valid data
- If a coherence message contains the owner token, it also contains valid data

Distributed Logical Time

- Each node has a time that is incremented on a message send (or receive) – the message carries the sender's timestamp with it
- If the receiver's time is less than the timestamp, the receiver updates its clock to timestamp + 1

Token Coherence Signature Checker

- Confirm that the {tokens received, timestamp} at the sender matches that at the receiver

- Over a given time interval, maintain a signature to represent all received/sent tokens/timestamps and confirm that they are consistent (allow a grace period as a message with an old timestamp may not have been delivered by the end of the interval – if the message is stuck for a really long time, treat it as a fault)

- If not, rely on previously proposed checkpoint mechanisms to rollback to valid state and re-execute

Signature

- Positive for a receive, negative for a send; the sum of signatures should total to zero
- $$TCS_{token}(B) = \sum_{t \in I} N_t \cdot (T+1)^t$$
- I is the time interval; t is every logical time step in I; N_t is the number of tokens received at time t; T is the total number of tokens for a block; can limit the size of the signature by making every computation modulo n
- Errors can also happen if the token is assigned to the wrong block; an address signature helps detect such errors:

$$TCS_{addr,owner} = [\Sigma A_t\, (max\{A\}+1)^t]\ mod\ n$$

Example

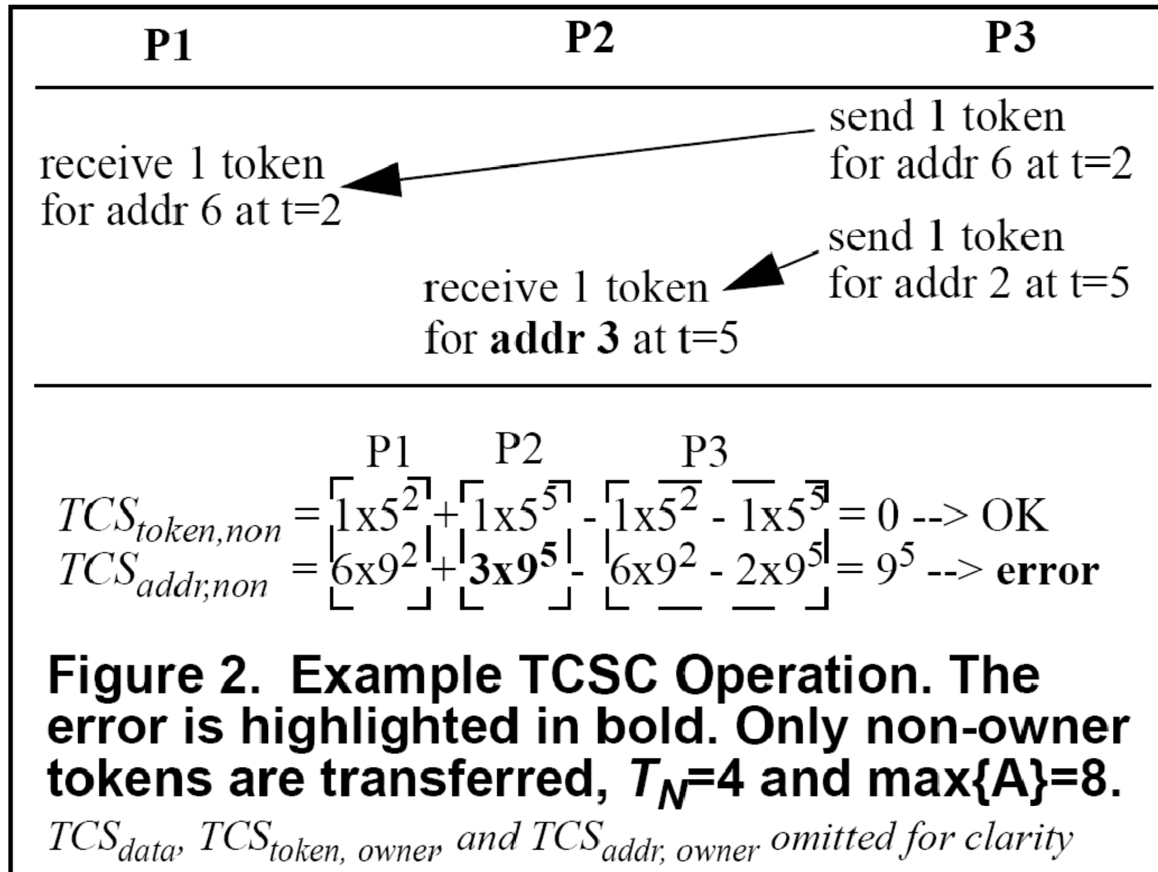


Figure 2. Example TCSC Operation. The error is highlighted in bold. Only non-owner tokens are transferred, T_N=4 and max{A}=8.

TCS_{data}, $TCS_{token, owner}$ and $TCS_{addr, owner}$ omitted for clarity

Results summary: bandwidth overhead (timestamps and signature collection) of less than 7% (negligible performance impact)

Title

- Bullet