ScalableBulk: Scalable Cache Coherence for Atomic Blocks in a Lazy Environmnet

Xuehai Qian, Wonsun Ahn, Josep Torrellas
University of Illinois

http://iacoma.cs.uiuc.edu/
Motivation
Motivation

- Architectures that continuously execute Atomic Blocks or Chunks (e.g., TCC, BulkSC)
Motivation

- Architectures that continuously execute **Atomic Blocks or Chunks** (e.g., TCC, BulkSC)
- Chunk: group of dynamically contiguous instructions executed atomically
Motivation

- Architectures that continuously execute Atomic Blocks or Chunks (e.g., TCC, BulkSC)
  - Chunk: group of dynamically contiguous instructions executed atomically
  - Provide performance and programmability advantages [Hammond 04], [Ahn 10]
Motivation

- Architectures that continuously execute **Atomic Blocks or Chunks** (e.g., TCC, BulkSC)
  - Chunk: group of dynamically contiguous instructions executed atomically
  - Provide performance and programmability advantages [Hammond 04], [Ahn 10]
  - An important operation is commit: makes the state of chunk visible atomically
Motivation

- Architectures that continuously execute Atomic Blocks or Chunks (e.g., TCC, BulkSC)
- Chunk: group of dynamically contiguous instructions executed atomically
- Provide performance and programmability advantages [Hammond 04], [Ahn 10]
- An important operation is commit: makes the state of chunk visible atomically
- Designs with lazy detection of chunk conflicts
Motivation

- Architectures that continuously execute **Atomic Blocks or Chunks** (e.g., TCC, BulkSC)
  - Chunk: group of dynamically contiguous instructions executed atomically
- Provide performance and programmability advantages [Hammond 04], [Ahn 10]
- An important operation is commit: makes the state of chunk visible atomically
- Designs with **lazy** detection of chunk conflicts
- Commit involves updating cache states and checking for conflicts
Motivation

- Architectures that continuously execute Atomic Blocks or Chunks (e.g., TCC, BulkSC)
  - Chunk: group of dynamically contiguous instructions executed atomically
  - Provide performance and programmability advantages [Hammond 04], [Ahn 10]
  - An important operation is commit: makes the state of chunk visible atomically
  - Designs with lazy detection of chunk conflicts
  - Commit involves updating cache states and checking for conflicts
  - In lazy directory-based cache coherent systems, commit is very challenging
Motivation

• Architectures that continuously execute Atomic Blocks or Chunks (e.g., TCC, BulkSC)
  • Chunk: group of dynamically contiguous instructions executed atomically
  • Provide performance and programmability advantages [Hammond 04], [Ahn 10]
  • An important operation is commit: makes the state of chunk visible atomically

• Designs with lazy detection of chunk conflicts
  • Commit involves updating cache states and checking for conflicts
  • In lazy directory-based cache coherent systems, commit is very challenging
    • Requires updating the states of the distributed caches in a way that appears that chunks execute in a total order
Motivation

- Architectures that continuously execute Atomic Blocks or Chunks (e.g., TCC, BulkSC)
  - Chunk: group of dynamically contiguous instructions executed atomically
  - Provide performance and programmability advantages [Hammond 04], [Ahn 10]
  - An important operation is commit: makes the state of chunk visible atomically
- Designs with lazy detection of chunk conflicts
  - Commit involves updating cache states and checking for conflicts
- In lazy directory-based cache coherent systems, commit is very challenging
  - Requires updating the states of the distributed caches in a way that appears that chunks execute in a total order
  - In large systems, it results in an execution bottleneck
Commit in Directory-Based Machine
Commit in Directory-Based Machine

Conventional Cache Coherence
Commit in Directory-Based Machine

Conventional Cache Coherence
Commit in Directory-Based Machine

Conventional Cache Coherence
Commit in Directory-Based Machine

Conventional Cache Coherence
Commit in Directory-Based Machine

Conventional Cache Coherence
Commit in Directory-Based Machine

Conventional Cache Coherence

Chunk-based Cache Coherence

Xuehai Qian

Scalable Bulk Protocol
Commit in Directory-Based Machine

Conventional Cache Coherence

Chunk-based Cache Coherence
Commit in Directory-Based Machine

Conventional Cache Coherence

Chunk-based Cache Coherence

Xuehai Qian
Scalable Bulk Protocol
Commit in Directory-Based Machine

Conventional Cache Coherence

Chunk-based Cache Coherence

Xuehai Qian

Scalable Bulk Protocol
Commit in Directory-Based Machine

Conventional Cache Coherence

Chunk-based Cache Coherence

Xuehai Qian
Scalable Bulk Protocol
Recent Commit Protocols
Recent Commit Protocols

• BulkSC [Ceze 07]:

Friday, March 4, 2011
Recent Commit Protocols

- BulkSC [Ceze 07]:
  - Centralized commit arbiter
Recent Commit Protocols

- BulkSC [Ceze 07]:
  - Centralized commit arbiter
- Scalable TCC [Chafi 07]:
Recent Commit Protocols

- BulkSC [Ceze 07]:
  - Centralized commit arbiter
- Scalable TCC [Chafi 07]:
  - First distributed scheme
Recent Commit Protocols

- BulkSC [Ceze 07]:
  - Centralized commit arbiter
- Scalable TCC [Chafi 07]:
  - First distributed scheme
  - Enforce total order by grabbing a commit token
Recent Commit Protocols

- BulkSC [Ceze 07]:
  - Centralized commit arbiter
- Scalable TCC [Chafi 07]:
  - First distributed scheme
  - Enforce total order by grabbing a commit token
  - Serialization and global communication
Recent Commit Protocols

- **BulkSC [Ceze 07]:**
  - Centralized commit arbiter

- **Scalable TCC [Chafi 07]:**
  - First distributed scheme
  - Enforce total order by grabbing a commit token
    - Serialization and global communication

- **SEQ Protocol [Pugsley 08]:**
Recent Commit Protocols

• BulkSC [Ceze 07]:
  • Centralized commit arbiter

• Scalable TCC [Chafi 07]:
  • First distributed scheme
  • Enforce total order by grabbing a commit token
    • Serialization and global communication

• SEQ Protocol [Pugsley 08]:
  • Extends Scalable TCC by eliminating global communication
Recent Commit Protocols

- BulkSC [Ceze 07]:
  - Centralized commit arbiter
- Scalable TCC [Chafi 07]:
  - First distributed scheme
  - Enforce total order by grabbing a commit token
    - Serialization and global communication
- SEQ Protocol [Pugsley 08]:
  - Extends Scalable TCC by eliminating global communication
  - Still requires serialization of commits that use the same directory module
SEQ Protocol
SEQ Protocol

P=Processor  D=Directory module

P0  P1

D0  D1  D2

Xuehai Qian

Scalable Bulk Protocol
SEQ Protocol

P=Processor    D=Directory module    Chunk 0

P0    P1

D0    D1    D2
SEQ Protocol

P=Processor    D=Directory module

P0    P1

D0    D1    D2

Chunk 0    Chunk 1
SEQ Protocol

P=Processor    D=Directory module

P0    P1

D0    D1    D2

Chunk 0    Chunk 1
SEQ Protocol

<table>
<thead>
<tr>
<th>P</th>
<th>D</th>
<th>newborn Chunk 0</th>
<th>Chunk 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>D0</td>
<td>Purple</td>
<td>Orange</td>
</tr>
<tr>
<td>P1</td>
<td>D1</td>
<td>Purple</td>
<td>Orange</td>
</tr>
<tr>
<td>P1</td>
<td>D2</td>
<td>Purple</td>
<td>Orange</td>
</tr>
</tbody>
</table>
SEQ Protocol

P=Processor    D=Directory module

Chunk 0
Chunk 1

P0
P1

P0
P1

D0
D1
D2

D0
D1
D2
SEQ Protocol

P = Processor    D = Directory module

P0          P1

D0          D1          D2

Chunk 0    Chunk 1

occ

D0          D1          D2

Xuehai Qian
Scalable Bulk Protocol
SEQ Protocol

P = Processor    D = Directory module

P0    P1

D0    D1    D2

Chunk 0

Chunk 1

occ
SEQ Protocol

P=Processor    D=Directory module

D0    D1    D2

P0    P1

D0    D1    D2

occ    ack

Chunk 0

Chunk 1

Xuehai Qian

Scalable Bulk Protocol
SEQ Protocol

P=Processor    D=Directory module

Chunk 0

Chunk 1

P0

P1

D0

D1

D2

occ

ack

occ

D0

D1

D2

Xuehai Qian

ScalableBulk Protocol
SEQ Protocol

P=Processor    D=Directory module

Chunk 0    Chunk 1

P0    P1

D0    D1    D2

occ    ack    occ

D0    D1    D2
SEQ Protocol

P=Processor    D=Directory module

Chunk 0

Chunk 1

occ

ack

occ

ack

P0    P1

D0    D1    D2

D0    D1    D2

Xuehai Qian

Scalable Bulk Protocol
SEQ Protocol

P=Processor    D=Directory module

P0
D0
P1
D1
D2

P0
P1

D0
D1
D2

Chunk 0

Chunk 1

occ
ack
occ
ack
SEQ Protocol

P=Processor    D=Directory module

P0          P1
D0          D1          D2

P0          P1
D0          D1          D2

occ
ack
occ

P0          P1
D0          D1          D2

Chunk 0

Chunk 1

c

Xuehai Qian
Scalable Bulk Protocol

Friday, March 4, 2011
SEQ Protocol

P = Processor    D = Directory module

Chunk 0    Chunk 1

P0    P1

D0    D1    D2

occ
ack

occ

P0    P1

D0    D1    D2

occ

occ

ack

occ

Xuehai Qian
Scalable Bulk Protocol
SEQ Protocol

P=Processor  D=Directory module

P0  P1  
D0  D1  D2

Chunk 0  Chunk 1

P0  P1
D0  D1  D2

occ

cannot overlap

occ

ack

occ

ack

occ

D0  D1  D2

Xuehai Qian

Scalable Bulk Protocol

Friday, March 4, 2011
SEQ Protocol

P=Processor  D=Directory module

P0  P1  D0  D1  D2

Chunk 0  Chunk 1

cannot overlap

occ
d
ack

occ
d
ack

occ
d
ack

occ
d
ack

occ
d
ack

occ
d
ack

occ
d
ack

occ
d
ack
SEQ Protocol

P=Processor    D=Directory module

P0    P1

D0    D1    D2

P0    P1

D0    D1    D2

occ

ack

occ

cannot overlap

Chunk 0    Chunk 1

Processing nodes:

D0    D1    D2

P0

P1

D0    D1    D2

P0    P1

D0    D1    D2

occ

ack
SEQ Protocol

P=Processor    D=Directory module

Chunk 0          Chunk 1

cannot overlap

Xuehai Qian

Scalable Bulk Protocol
SEQ Protocol

P = Processor    D = Directory module

Chunk 0

Chunk 1

cannot overlap

P0

P1

D0

D1

D2

occ

occ

ack

ack
SEQ Protocol

P=Processor    D=Directory module

P0      P1
D0    D1    D2

Chunk 0

P0      P1
D0    D1    D2

cannot overlap

Xuehai Qian

Scalable Bulk Protocol
SEQ Protocol

P=Processor    D=Directory module

P0
D0

P1
D1

D2

P0

P1

occ

ack

occ

ack

D0

D1

D2

P0

P1

occ

D0

D1

D2

D0

D1

D2

cannot overlap

Chunk 0

Chunk 1

Xuehai Qian

Scalable Bulk Protocol
SEQ Protocol

P=Processor    D=Directory module

P0          P1
D0          D1          D2

P0
D0          D1          D2

occ
ack

P0          P1
D0          D1          D2

P0
D0          D1          D2

cannot overlap
SEQ Protocol

P=Processor    D=Directory module

P0
D0
P1
D1
D2

P=Processor    D=Directory module

Chunk 0

Chunk 1
cannot overlap

P0
P1
D0
D1
D2

P0
P1
D0
D1
D2

occ

ack

occ

ack
SEQ Protocol

P = Processor    D = Directory module

Chunk 0

Cannot overlap

P0

P1

D0

D1

D2

P0

P1

D0

D1

D2

occ

ack

occ

ack

occ

ack

occ

ack
SEQ Protocol

P=Processor    D=Directory module

P0  P1

D0  D1  D2

Chunk 0

Chunk 1

cannot overlap

Commits of chunk 0 and chunk 1 are serialized.
Commits that use the same directory module are serialized even when they touch non-overlapped lines.

Commits of chunk 0 and chunk 1 are serialized.

cannot overlap
Outline

• Motivation
• ScalableBulk
• Evaluation
Goals for Scalable Commit
Goals for Scalable Commit

• No centralized structure
Goals for Scalable Commit

- No centralized structure
- Committing processor communicates only with the relevant directory modules
Goals for Scalable Commit

- No centralized structure
- Committing processor communicates only with the relevant directory modules
- Allow concurrent commits of chunks that use the same directory module, as long as the accessed addresses do not overlap
Contribution: ScalableBulk
**Contribution: ScalableBulk**

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system
Contribution: ScalableBulk

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system
- Key properties:
Contribution: ScalableBulk

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system
- Key properties:
  - Enable multiple *concurrent* chunk commits that use the same directory module
Contribution: ScalableBulk

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system

- Key properties:
  - Enable multiple **concurrent** chunk commits that use the **same directory module**
  - Made possible by integrating **signatures** into directory design
Contribution: ScalableBulk

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system

- Key properties:
  - Enable multiple **concurrent** chunk commits that use the **same directory module**
  - Made possible by integrating **signatures** into directory design
  - Better tolerance to commits that use many directories
Contribution: ScalableBulk

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system

- Key properties:
  - Enable multiple *concurrent* chunk commits that use the same directory module
  - Made possible by integrating *signatures* into directory design
  - Better tolerance to commits that use many directories
  - *Eliminates* all centralized structures and global communications
Contribution: ScalableBulk

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system

- **Key properties:**
  - Enable multiple *concurrent* chunk commits that use the same directory module
  - Made possible by integrating *signatures* into directory design
  - Better tolerance to commits that use many directories
  - *Eliminates* all centralized structures and global communications
  - Committing processor only communicates with the *relevant* directories (the homes of the addresses accessed by the chunk)
Contribution: ScalableBulk

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system

- **Key properties:**
  - Enable multiple **concurrent** chunk commits that use the **same directory module**
  - Made possible by integrating **signatures** into directory design
  - Better tolerance to commits that use many directories
  - **Eliminates** all centralized structures and global communications
  - Committing processor only communicates with the **relevant directories** (the homes of the addresses accessed by the chunk)
  - More **scalable** than previous schemes
Contribution: ScalableBulk

- **ScalableBulk**: protocol for bottleneck-free commit of chunks in directory-based system

- **Key properties:**
  - Enable multiple **concurrent** chunk commits that use the **same directory module**
  - Made possible by integrating **signatures** into directory design
  - Better tolerance to commits that use many directories
  - Eliminates all centralized structures and global communications
    - Committing processor only communicates with the **relevant directories** (the homes of the addresses accessed by the chunk)
  - More **scalable** than previous schemes

- **Results**: practically eliminates all commit stall overhead for 64 processors
ScalableBulk Protocol Primitives
ScalableBulk Protocol Primitives

• Allowing multiple non-overlapping commits to use the same directory module
ScalableBulk Protocol Primitives

- Allowing multiple non-overlapping commits to use the same directory module
- Grouping directory modules
ScalableBulk Protocol Primitives

- Allowing multiple non-overlapping commits to use the same directory module
- Grouping directory modules
- Initiating the commit optimistically
ScalableBulk Protocol Primitives

- Allowing multiple non-overlapping commits to use the same directory module
- Grouping directory modules
- Initiating the commit optimistically

Many-Core Architecture Considered
Primitive 1:
Allowing Concurrent Non-overlapping Commits
Primitive 1:
Allowing Concurrent Non-overlapping Commits

- Read and write footprint of a chunk is summarized in read and write signatures using Bloom filters
Primitive 1: Allowing Concurrent Non-overlapping Commits

- Read and write footprint of a chunk is summarized in read and write signatures using Bloom filters
- On commit: signatures are sent to the relevant directory modules
Primitive 1: Allowing Concurrent Non-overlapping Commits

- Read and write footprint of a chunk is summarized in read and write signatures using Bloom filters
- On commit: signatures are sent to the relevant directory modules
- Only the addresses in the signature are locked in the directory during the commit
Primitive 1: Allowing Concurrent Non-overlapping Commits

- Read and write footprint of a chunk is summarized in read and write signatures using Bloom filters
- On commit: signatures are sent to the relevant directory modules
- Only the addresses in the signature are locked in the directory during the commit
- Other chunks can commit using the same directory if their signatures do not conflict with the committing signatures
Primitive 1: Allowing Concurrent Non-overlapping Commits
Primitive 1:
Allowing Concurrent Non-overlapping Commits

Currently Committing Signatures

R2 \( \bigcap \) \( \bigcap \) \( \bigcap \) \( \bigcap \)
\( \bigcap \)

W2

Incoming Signatures

W1

W0

if not \( \emptyset \): Nack commit
else: Start committing

NOR
Primitive 1: Allowing Concurrent Non-overlapping Commits

- Enables more concurrent commits

![Diagram](image)

Currently Committing Signatures

W0 → W1

R2 → W2

Incoming Signatures

if not ∅: Nack commit
else: Start committing

Xuehai Qian
Scalable Bulk Protocol

Friday, March 4, 2011
Primitive 2:
Grouping Directory Modules
Primitive 2: Grouping Directory Modules

- On a chunk commit: the relevant directory modules
Primitive 2: Grouping Directory Modules

- On a chunk commit: the relevant directory modules
- Coordinate their transitions by exchanging messages
Primitive 2:
Grouping Directory Modules

• On a chunk commit: the relevant directory modules
• Coordinate their transitions by exchanging messages
• Form a Directory Group
Primitive 2: Grouping Directory Modules

- On a chunk commit: the relevant directory modules
- Coordinate their transitions by exchanging messages
- Form a Directory Group
- Identify a leader module that sends messages to the caches and the committing processor on behalf of the group
Primitive 2: Grouping Directory Modules

- On a chunk commit: the relevant directory modules
- Coordinate their transitions by exchanging messages
- Form a Directory Group
- Identify a leader module that sends messages to the caches and the committing processor on behalf of the group

- Grouping Protocol:
Primitive 2: Grouping Directory Modules

- On a chunk commit: the relevant directory modules
- Coordinate their transitions by exchanging messages
- Form a Directory Group
- Identify a leader module that sends messages to the caches and the committing processor on behalf of the group

- Grouping Protocol:
  - Complete distributed operation
Primitive 2: Grouping Directory Modules

- On a chunk commit: the relevant directory modules
- Coordinate their transitions by exchanging messages
- Form a Directory Group
- Identify a leader module that sends messages to the caches and the committing processor on behalf of the group

- Grouping Protocol:
  - Complete distributed operation
  - Few messages are required
Primitive 2: Grouping Directory Modules

- On a chunk commit: the relevant directory modules
- Coordinate their transitions by exchanging messages
- Form a Directory Group
- Identify a leader module that sends messages to the caches and the committing processor on behalf of the group

- Grouping Protocol:
  - Complete distributed operation
  - Few messages are required
  - Leader: lowest-numbered directory module in the group
Primitive 2: Grouping Directory Modules
Primitive 2: Grouping Directory Modules

P=Processor  D=Directory module

P0

D0  D1  D2
Primitive 2: Grouping Directory Modules

P = Processor
D = Directory module

Chunk 0

P0

D0  D1  D2
Primitive 2: Grouping Directory Modules

P=Processor    D=Directory module    Chunk 0

Diagram:
- P0
- D0
- D1
- D2

Friday, March 4, 2011
Primitive 2: Grouping Directory Modules

P=Processor  D=Directory module  Chunk 0

Leader

D0  D1  D2

P0

Xuehai Qian
Scalable Bulk Protocol
Primitive 2:
Grouping Directory Modules

P=Processor  D=Directory module  Chunk 0

Leader

P0

D0  D1  D2

P0

D0  D1  D2
Primitive 2: Grouping Directory Modules

P=Processor  D=Directory module  Chunk 0

Leader

Sig{R,W}  Sig{R,W}  Sig{R,W}

D0  D1  D2

D0  D1  D2
Primitive 2: Grouping Directory Modules

P = Processor  D = Directory module

Chunk 0

Leader

Sig{R,W}  Sig{R,W}  Sig{R,W}

D0  D1  D2

D0  D1  D2
Primitive 2: Grouping Directory Modules

P=Processor    D=Directory module    Chunk 0

Leader

Sig{R,W}    Sig{R,W}    Sig{R,W}

D0  D1  D2

D0  D1  D2
Primitive 2: Grouping Directory Modules

P = Processor   D = Directory module

Chunk 0

Leader

P0

D0  D1  D2

Sig{R,W} Sig{R,W} Sig{R,W}

D0 grab D1  D2

Xuehai Qian

Scalable Bulk Protocol
Primitive 2: Grouping Directory Modules

P=Processor    D=Directory module    Chunk 0

Leader

D0 D1 D2

Sig{R,W}  Sig{R,W}  Sig{R,W}

D0 grab  D1  D2

Xuehai Qian
Scalable Bulk Protocol
Friday, March 4, 2011
Primitive 2:
Grouping Directory Modules

P=Processor    D=Directory module

Chunk 0

Leader

Sig{R,W}  Sig{R,W}  Sig{R,W}

D0 grab   D1 grab   D2
Primitive 2: Grouping Directory Modules

P=Processor  D=Directory module  Chunk 0

Leader

P0

D0  D1  D2

Sig{R,W}  Sig{R,W}  Sig{R,W}

D0 grab  D1 grab  D2
Primitive 2: Grouping Directory Modules

P=Processor    D=Directory module    Chunk 0

P0

D0    D1    D2

Sig{R,W}    Sig{R,W}    Sig{R,W}

D0 grab    D1 grab    D2 grab
Primitive 2:
Grouping Directory Modules

P = Processor  D = Directory module  Chunk 0

Leader

Group is formed

Sig{R,W}  Sig{R,W}  Sig{R,W}

D0  D1  D2

D0 grab  D1 grab  D2 grab
Primitive 2: Grouping Directory Modules

P = Processor  D = Directory module

Chunk 0

Leader

Group is formed

Sig{R,W}

Sig{R,W}

Sig{R,W}

D0 grab  D1 grab  D2 grab
Primitive 2: Grouping Directory Modules

P = Processor    D = Directory module

Chunk 0

Leader

Group is formed

Sig{R,W}

Sig{R,W}

Sig{R,W}

D0 grab

D1 grab

D2 grab

P0

D0

D1

D2

commit_ack

P0

D0

D1

D2

Xuehai Qian

Scalable Bulk Protocol
Primitive 2: Grouping Directory Modules

P=Processor  D=Directory module

Chunk 0

Group is formed

Leader

Sig{R,W}

Sig{R,W}

Sig{R,W}

P0

D0  D1  D2

P0

D0  D1  D2

D0 grab  D1 grab  D2 grab

commit_ack

Xuehai Qian  Scalable Bulk Protocol
Primitive 2: Grouping Directory Modules

P=Processor  D=Directory module

<table>
<thead>
<tr>
<th>P0</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
</tr>
</thead>
</table>

Chunk 0

Leader

Group is formed

Sig\{R,W\}

D0 grab  D1 grab  D2 grab

Sig\{R,W\}  Sig\{R,W\}  Sig\{R,W\}

commit_ack

P0

succ

D0  D1  D2

succ

Xuehai Qian

Scalable Bulk Protocol
Primitive 2: Grouping Directory Modules

P=Processor    D=Directory module

Group is formed

Leader

Commit_ack

Bulk_inv

Succ

Sig{R,W}
Primitive 2: Grouping Directory Modules

P = Processor    D = Directory module

Chunk 0

Leader

Group is formed

Sig{R, W}

Sig{R, W}

Sig{R, W}

D0 grab    D1 grab    D2 grab

commit_ack    bulk_inv

D0 succ    D1 succ    D2 succ

Xuehai Qian

Scalable Bulk Protocol
Primitive 2: Grouping Directory Modules

P = Processor  D = Directory module

Chunk 0

Leader

Group is formed

Sig{R,W} Sig{R,W} Sig{R,W}

D0 grab  D1 grab  D2 grab

commit_ack

bulk_inv

succ

inv_ack

succ

D0  D1  D2
Primitive 2: Grouping Directory Modules

P=Processor    D=Directory module

Chunk 0

Leader

Group is formed

Sig{R,W}

Sig{R,W}

Sig{R,W}

D0 grab

D1 grab

D2 grab

P0

D0

D1

D2

commit_ack

bulk_inv

succ

inv_ack

Leader

Commit_ack

Bulk_inv

Succ

Inv_ack

Xuehai Qian

Scalable Bulk Protocol
Primitive 2: Grouping Directory Modules

P = Processor  D = Directory module

Chunk 0

Leader

Group is formed

Sig{R,W}

D0 grab  D1 grab  D2 grab

Sig{R,W}

Sig{R,W}

P0

D0
D1
D2

P0

commit_ack

bulk_inv

P0

succ

D0
D1
D2

P0

inv_ack

done

D0
D1
D2

P0

done

D0
D1
D2

Xuehai Qian

Scalable Bulk Protocol

Friday, March 4, 2011
**Primitive 2:**
**Grouping Directory Modules**

P = Processor  
D = Directory module  
Chunk 0

Leader

Group is formed

Sig{R,W}  
Sig{R,W}  
Sig{R,W}

commit_ack  
bulk_inv

succ  
inv_ack

done  
done
Primitive 2: Grouping Directory Modules

P=Processor    D=Directory module

Leader

Group is formed

Sig{R,W}

D0 grab    D1 grab    D2 grab

D0 grab    D1 grab    D2 grab

Sig{R,W}

Commit finished

P0

D0

D1

D2

Commit finished

P0

D0

D1

D2

Sig{R,W}

Sig{R,W}

Sig{R,W}

commit_ack

bulk_inv

succ

inv_ack

done

D0

D1

D2

D0

D1

D2

Xuehai Qian

Scalable Bulk Protocol
Distributed Conflict Detection
Distributed Conflict Detection

P=Processor    D=Directory module

P0    P1    D0    D1    D2
Distributed Conflict Detection

P=Processor  D=Directory module  Chunk 0

P0  P1

D0  D1  D2

Xuehai Qian

Scalable Bulk Protocol
Distributed Conflict Detection

P=Processor    D=Directory module

P0    P1

D0    D1    D2

Chunk 0
Distributed Conflict Detection

P=Processor    D=Directory module

P0    P1

D0    D1    D2

Chunk 0    Chunk 1
Distributed Conflict Detection

P=Processor    D=Directory module

P0     P1

D0     D1     D2

Chunk 0     Chunk 1

Friday, March 4, 2011
Distributed Conflict Detection

P=Processor        D=Directory module       Chunk 0

P0          P1

D0  D1  D2

---

Xuehai Qian

Scalable Bulk Protocol
Distributed Conflict Detection

P=Processor    D=Directory module

P0
D0
D1
D2

P1

Sig{R,W}

Chunk 0

Chunk 1

Friday, March 4, 2011
Distributed Conflict Detection

P=Processor  D=Directory module

Chunk 0  Chunk 1

P0  P1

D0  D1  D2

D0  D1  D2

Sig{R,W}  Sig{R,W}
Distributed Conflict Detection

P = Processor    D = Directory module

Chunk 0    Chunk 1

D0    D1    D2

D0    D1    D2

Sig{R,W}    Sig{R,W}
Distributed Conflict Detection

P=Processor  D=Directory module  Chunk 0  Chunk 1

P0  P1

D0  D1  D2

Sig\{R,W\}  Sig\{R,W\}

D0  D1  D2

grab
Distributed Conflict Detection

P=Processor    D=Directory module

 Chunk 0

 Chunk 1

P0
D0

P1
D1

Sig\{R,W\}
Sig\{R,W\}

D2

D1

grab

D2

P0

P1
Distributed Conflict Detection

P=Processor    D=Directory module

P0    P1

D0    D1    D2

Chunk 0    Chunk 1

Sig{R,W}  Sig{R,W}

grab  grab  grab

Xuehai Qian  Scalable Bulk Protocol
Distributed Conflict Detection

P = Processor  D = Directory module

P0  P1  D0  D1  D2

Chunk 0  Chunk 1

P0  P1

D0  D1  D2

Sig{R,W}  Sig{R,W}

grab  grab  grab

Friday, March 4, 2011
Distributed Conflict Detection

P = Processor    D = Directory module

Chunk 0

P0

D0

P1

D1

D2

Chunk 1

P0

D0

P1

D1

D2

Sig{R,W}

Sig{R,W}

grab

grab

grab

Xuehai Qian

Scalable Bulk Protocol
Distributed Conflict Detection

P=Processor    D=Directory module

P0          P1

D0        D1        D2

Chunk 0    Chunk 1

P0          P1

D0        D1        D2

Sig{R,W}    Sig{R,W}

D0        D1        D2

grab

grab

Xuehai Qian

Scalable Bulk Protocol
Distributed Conflict Detection

P=Processor   D=Directory module

P0   P1
D0   D1   D2

Sig{R,W}  Sig{R,W}  grab  grab
D0   D1   D2

Chunk 0   Chunk 1

P0   P1
D0   D1   D2

grab

Xuehai Qian

Scalable Bulk Protocol

Friday, March 4, 2011
Distributed Conflict Detection

P=Processor    D=Directory module

chunk 0

chunk 1

P0  P1

D0  D1  D2

P0  P1

D0  D1  D2

sig{R,W}

grab

Sig{R,W}

grab

grab
Distributed Conflict Detection

P=Processor    D=Directory module

Chunk 0

Chunk 1

P=Processor    D=Directory module

Sig{R,W}

Sig{R,W}

grab

grab

grab

Xuehai Qian

Scalable Bulk Protocol
Distributed Conflict Detection

P=Processor    D=Directory module

D0  D1  D2

P0  P1

D0  D1  D2

P0  P1  P0  P1

Conflicts are detected in D1

Xuehai Qian  Scalable Bulk Protocol
Distributed Conflict Detection

P=Processor  D=Directory module

P0  P1  D0  D1  D2

P0  P1  D0  D1  D2

Sig{R,W}  Sig{R,W}  grab  grab

Conflict is detected in D1
Distributed Conflict Detection

P=Processor    D=Directory module

Chunk 0

Chunk 1

P0
P1

D0
D1
D2

Sig{R,W}

D0 grab
D1 grab
D2

P0
P1

D0
D1
D2

g_failure

commit failure

Conflicts detected in D1

Xuehai Qian
Scalable Bulk Protocol

Friday, March 4, 2011
Distributed Conflict Detection

P=Processor  D=Directory module

chunk 0  chunk 1

P0  P1  P0  P1

D0  D1  D2  D0  D1  D2

Sig{R,W}  Sig{R,W}  g_failure

grab  grab  commit failure

Conflict is detected in D1

Xuehai Qian  Scalable Bulk Protocol
Primitive 2:
Grouping Directory Modules

Friday, March 4, 2011
Primitive 2: 
Grouping Directory Modules

• Deadlock is avoided by following a fixed directory-module traversal order
Primitive 2: Grouping Directory Modules

- Deadlock is avoided by following a fixed directory-module traversal order
- Multiple groups can commit concurrently
Concurrent Commit
Concurrent Commit

P=Processor    D=Directory module

P0  P1

D0  D1  D2

Xuehai Qian
Scalable Bulk Protocol
Concurrent Commit

P=Processor D=Directory module

P0 = P1

D0 D1 D2

Chunk 0
Concurrent Commit

P=Processor    D=Directory module

P0    P1

D0    D1    D2

Chunk 0

Xuehai Qian

Scalable Bulk Protocol

Friday, March 4, 2011
Concurrent Commit

P=Processor    D=Directory module

P0

D0

D1

D2

P1

Chunk 0

Chunk 1

Friday, March 4, 2011
Concurrent Commit

P=Processor  D=Directory module

P0  Chunk 0

P1  Chunk 1

D0
D1
D2
Concurrent Commit

P=Processor  D=Directory module

P0  P1

D0  D1  D2

Chunk 0  Chunk 1

Xuehai Qian

Scalable Bulk Protocol
Concurrent Commit

P=Processor  D=Directory module

\[ \begin{align*}
P_0 & \quad D_0 \\
P_1 & \quad D_1 \\
\end{align*} \]

\[ \begin{align*}
P_0 & \quad D_0 \\
P_1 & \quad D_1 \\
\end{align*} \]

\[ \begin{align*}
\text{Sig\{R,W\}} & \quad \text{Sig\{R,W\}} \\
D_0 & \quad D_1 \\
\end{align*} \]

Chunk 0  Chunk 1
Concurrent Commit

P=Processor    D=Directory module

P0
D0

P1
D1

D2

Sig{R,W}

Chunk 0

Chunk 1

Sig{R,W}
Concurrent Commit

\[ P=\text{Processor} \quad D=\text{Directory module} \]

\[ \text{Chunk 0} \quad \text{Chunk 1} \]

\[ P_0 \quad P_1 \]

\[ D_0 \quad D_1 \quad D_2 \]

\[ \text{Sig}\{R,W\} \quad \text{Sig}\{R,W\} \]

Friday, March 4, 2011
Concurrent Commit

P=Processor    D=Directory module

\[
\begin{aligned}
P0 & \rightarrow D0 \\
P1 & \rightarrow D1 \\
\end{aligned}
\]

\[
\begin{aligned}
P0 & \rightarrow D0 \\
P1 & \rightarrow D1 \\
D2 & \rightarrow D1 \\
\end{aligned}
\]

Sig\{R,W\}  Sig\{R,W\}  grab

Chunk 0  Chunk 1

Xuehai Qian  Scalable Bulk Protocol
Concurrent Commit

P=Processor  D=Directory module

 Chunk 0  Chunk 1

P0  P1

D0  D1  D2

Sig{R,W}  Sig{R,W}

D0  D1  D2

grab
Concurrent Commit

\[ P = \text{Processor} \quad D = \text{Directory module} \quad \quad \begin{array}{c} \text{Chunk 0} \\ \text{Chunk 1} \end{array} \]

Diagram:

- Two processors: P0 and P1
- Three directory modules: D0, D1, D2
- Connections:
  - P0 to D0 and D1
  - P1 to D1 and D2
  - D0 to P0
  - D1 to P1
  - D2 to P1

- Symbols:
  - Sig\{R,W\}
  - grab

Friday, March 4, 2011
Concurrent Commit

P=Processor  D=Directory module

Chunk 0  Chunk 1

P0  P1  D0  D1  D2  P0  P1  D0  D1  D2

Sig{R,W}  Sig{R,W}  grab  grab

D0  D1  D2  D0  D1  D2

Xuehai Qian

Scalable Bulk Protocol
Concurrent Commit

P = Processor  D = Directory module

P0  P1
D0  D1  D2

Sig{R,W}  Sig{R,W}
D0  D1  D2

grab  grab
Concurrent Commit

P = Processor    D = Directory module

P0  P1

D0  D1  D2

Chunk 0

P0  P1

D0  D1  D2

Chunk 1

Sig{R,W}

grab

Sig{R,W}

grab

grab

Xuehai Qian

Scalable Bulk Protocol

Friday, March 4, 2011
Concurrent Commit

P=Processor    D=Directory module

D0 D1 D2

P0 P1

Sig{R,W}  Sig{R,W}

D0 D1 D2

grab grab

Chunk 0 Chunk 1

P0 P1

D0 D1 D2

grab

Friday, March 4, 2011
Concurrent Commit

P=Processor  D=Directory module

D0  D1  D2

P0  P1

P=Processor  D=Directory module

Chunk 0

Chunk 1

P0  P1

D0  D1  D2

Sig{R,W}  Sig{R,W}

grab

grab

grab

P0  P1

D0  D1  D2

D0  D1  D2
Concurrent Commit

P=Processor    D=Directory module

D0    D1    D2

P0

D0

D1

D2

Sig{R,W}

grab

Sig{R,W}

grab

Chunk 0

Chunk 1

P=Processor    D=Directory module

D0    D1    D2

P0

D0

D1

D2

grab
Concurrent Commit

P=Processor    D=Directory module

D0 D1 D2

P0 P1

D0 D1 D2

Sig{R,W} grab

No conflict is detected in D1.

P0

Sig{R,W} grab

P1

P0

P1
Concurrent Commit

P=Processor    D=Directory module

P0  P1

D0  D1  D2

P0  P1

D0  D1  D2

No conflict is detected in D1.
Concurrent Commit

P=Processor  D=Directory module

Chunk 0  Chunk 1

P0  P1

D0  D1  D2

No conflict is detected in D1.
Concurrent Commit

P=Processor  D=Directory module

Chunk 0  Chunk 1

P0  P1

D0  D1  D2

No conflict is detected in D1.

Two groups commit concurrently.

Scalable Bulk Protocol
Primitive 3: Optimistic Commit Initiation
Primitive 3:
Optimistic Commit Initiation

• Idea:
Primitive 3: Optimistic Commit Initiation

• Idea:
  • Committing processor (CP) assumes its commit transaction will succeed
Primitive 3: Optimistic Commit Initiation

- Idea:
  - Committing processor (CP) assumes its commit transaction will succeed
  - CP consumes incoming messages before receiving OK to commit
Primitive 3: Optimistic Commit Initiation

- Idea:
  - Committing processor (CP) assumes its commit transaction will succeed
  - CP consumes incoming messages before receiving OK to commit
- Advantages: it enables more overlapping of commits
Primitive 3: Optimistic Commit Initiation

(a) Processor P0 and P1 with R0W0 and R1W1 operations.

(b) Commit success for group G0.

(c) NACK from P1 to P0.

(d) Commit failure and squashing.

Scalable Bulk Protocol
Directory Design

input buffer

output buffer

CST

confirmed?
hold?
leader?

C_Tag | Sigs | Chunk State | inval_vec | g_vec | l | h | c

ScalableBulk Protocol Engine
# Messages

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Description</th>
<th>Format</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit.request</td>
<td>Processor requests to commit a chunk. Message is sent to all the directory</td>
<td>$C.Tag, W.Sig, R.Sig, g.vec$</td>
<td>Proc $\rightarrow$ Dir(s)</td>
</tr>
<tr>
<td></td>
<td>modules in the read- and write-sets of the chunk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$g$ (or grab)</td>
<td>Source directory module is part of a group, and tries to grab the destination</td>
<td>$C.Tag, inval.vec$</td>
<td>Dir $\rightarrow$ Dir</td>
</tr>
<tr>
<td></td>
<td>module to put it into the same group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$g$.failure</td>
<td>A module detects that group formation has failed and notifies of the failure</td>
<td>$C.Tag$</td>
<td>Dir $\rightarrow$ Dir(s)</td>
</tr>
<tr>
<td></td>
<td>to all the modules in the group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$g$.success</td>
<td>The leader informs all the modules in the group that the group has been</td>
<td>$C.Tag$</td>
<td>Dir $\rightarrow$ Dir(s)</td>
</tr>
<tr>
<td></td>
<td>successfully formed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>commit.failure</td>
<td>The leader notifies the commit-requesting processor that the commit failed</td>
<td>$C.Tag$</td>
<td>Dir $\rightarrow$ Proc</td>
</tr>
<tr>
<td>commit.success</td>
<td>The leader notifies the commit-requesting processor that the commit is</td>
<td>$C.Tag$</td>
<td>Dir $\rightarrow$ Proc</td>
</tr>
<tr>
<td></td>
<td>successful</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bulk.inv</td>
<td>The leader sends out a bulk invalidation to all the sharer processors</td>
<td>$C.Tag, W.Sig$</td>
<td>Dir $\rightarrow$ Proc(s)</td>
</tr>
<tr>
<td>bulk.inv.ack</td>
<td>The leader receives a bulk invalidation acknowledgment from a sharer</td>
<td>$C.Tag$</td>
<td>Proc $\rightarrow$ Dir</td>
</tr>
<tr>
<td></td>
<td>processor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>commit.done</td>
<td>The leader releases all the modules in the group and requests the</td>
<td>$C.Tag$</td>
<td>Dir $\rightarrow$ Dir(s)</td>
</tr>
<tr>
<td></td>
<td>deallocation of the signatures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>commit.recall</td>
<td>A processor with a squashed chunk notifies the leader module of the squash.</td>
<td>$C.Tag, Dir.ID$</td>
<td>Proc $\rightarrow$ Dir,</td>
</tr>
<tr>
<td></td>
<td>The message is piggy-backed on $bulk.inv.ack$ and $commit.done$ messages</td>
<td></td>
<td>Dir $\rightarrow$ Dir</td>
</tr>
</tbody>
</table>
Summary: Scalable Commit
Summary: Scalable Commit

- Commit has no centralization point
Summary: Scalable Commit

- Commit has no centralization point
- Committing processor communicates only with relevant directory modules (no message broadcasting)
Summary: Scalable Commit

- Commit has no centralization point
- Committing processor communicates only with relevant directory modules (no message broadcasting)
- Multiple committing chunks can use the same directory modules, if the addresses that they access do not overlap
Summary: Scalable Commit

- Commit has no centralization point
- Committing processor communicates only with relevant directory modules (no message broadcasting)
- Multiple committing chunks can use the same directory modules, if the addresses that they access do not overlap
- Similar to how conventional protocols support concurrent writes
Summary: Scalable Commit

- Commit has no centralization point
- Committing processor communicates only with relevant directory modules (no message broadcasting)
- Multiple committing chunks can use the same directory modules, if the addresses that they access do not overlap
- Similar to how conventional protocols support concurrent writes
- Optimistic Commit Initiation removes operations from critical path of the commit
Summary: Scalable Commit

• Commit has no centralization point
• Committing processor communicates only with relevant directory modules (no message broadcasting)
• Multiple committing chunks can use the same directory modules, if the addresses that they access do not overlap
• Similar to how conventional protocols support concurrent writes
• Optimistic Commit Initiation removes operations from critical path of the commit

Commit is truly scalable
Outline

• Motivation
• ScalableBulk
• Evaluation
Evaluation
Evaluation

- Cycle-accurate execution-driven simulator based on SESC and Pin
Evaluation

- Cycle-accurate execution-driven simulator based on SESC and Pin
- Number of cores: 32 and 64
Evaluation

• Cycle-accurate execution-driven simulator based on SESC and Pin
• Number of cores: 32 and 64
• 11 SPLASH-2 and 7 PARSEC applications
Evaluation

• Cycle-accurate execution-driven simulator based on SESC and Pin
• Number of cores: 32 and 64
• 11 SPLASH-2 and 7 PARSEC applications
• Implemented all existing protocols:
Evaluation

- Cycle-accurate execution-driven simulator based on SESC and Pin
- Number of cores: 32 and 64
- 11 SPLASH-2 and 7 PARSEC applications
- Implemented all existing protocols:
  - ScalableBulk
Evaluation

- Cycle-accurate execution-driven simulator based on SESC and Pin
- Number of cores: 32 and 64
- 11 SPLASH-2 and 7 PARSEC applications
- Implemented all existing protocols:
  - ScalableBulk
  - Scalable TCC
Evaluation

- Cycle-accurate execution-driven simulator based on SESC and Pin
- Number of cores: 32 and 64
- 11 SPLASH-2 and 7 PARSEC applications
- Implemented all existing protocols:
  - ScalableBulk
  - Scalable TCC
  - SEQ
Evaluation

- Cycle-accurate execution-driven simulator based on SESC and Pin
- Number of cores: 32 and 64
- 11 SPLASH-2 and 7 PARSEC applications
- Implemented all existing protocols:
  - ScalableBulk
  - Scalable TCC
  - SEQ
  - BulkSC
Execution Time

The diagram above shows the execution time for various benchmarks: Vips 32, Vips 64, Swaptions 32, Swaptions 64, Blackscholes 32, Blackscholes 64, Fluidanimate 32, Fluidanimate 64, Canneal 32, Canneal 64, Dedup 32, Dedup 64, Facesim 32, Facesim 64, and AVERAGE 32, AVERAGE 64. The execution times are presented for two different sizes: 32 and 64, with execution time values ranging from 0 to 0.1.

Key performance parameters include:
- **Squash**
- **Commit**
- **Cache Miss**
- **Useful**

The diagram effectively compares the execution times across these benchmarks and sizes, highlighting performance metrics such as Squash, Commit, Cache Miss, and Useful time.
• ScalableBulk practically eliminates all commit stall time

Friday, March 4, 2011
• ScalableBulk practically eliminates all commit stall time
• Other existing protocols suffer commit stall (see paper)
Directories Used Per Commit

Number of Directories

- Vips 32
- Vips 64
- Swaptions 32
- Swaptions 64
- Blackscholes 32
- Blackscholes 64
- Fluidanimate 32
- Fluidanimate 64
- Canreal 32
- Canreal 64
- Dedup 32
- Dedup 64
- Facesim 32
- Facesim 64
- AVERAGE 32
- AVERAGE 64

Friday, March 4, 2011
• Chunk commits use about 6 directories on average
Directories Used Per Commit

- Chunk commits use about 6 directories on average
- ScalableBulk is able to overlap the commits that use the same directories if the signatures do not conflict
Network Message Characterization

Number of Messages (%)

Vips  Swaptions  Blackscholes  Fluidanimate  Canneal  Dedup  Facesim  AVERAGE

S  B  S  S  Q  S  B  S  Q  Q  S  B  S  S  B  B

0  20  40  60  80  100
Network Message Characterization

![Bar chart showing network message characterization for different applications. The x-axis represents various applications such as Vips, Swaptions, Blackscholes, Fluidanimate, Canneal, Dedup, Facesim, and Average. The y-axis represents the number of messages as a percentage. The chart highlights different protocols such as Scalable TCC, ScalableBulk, SEQ, and BulkSC.](chart.png)
Network Message Characterization

• ScalableBulk sends fewer messages than other distributed protocols
Also in the paper
Also in the paper

- Mechanism to handle fairness and avoid starvation
Also in the paper

- Mechanism to handle fairness and avoid starvation
- Many details on the implementation of the ScalableBulk protocol
Also in the paper

- Mechanism to handle fairness and avoid starvation
- Many details on the implementation of the ScalableBulk protocol
- Detailed results characterizing various aspects of the protocol
Conclusion
Conclusion

- Proposed **ScalableBulk**: protocol for bottleneck-free commit of chunks in lazy directory-based system
Conclusion

• Proposed ScalableBulk: protocol for bottleneck-free commit of chunks in lazy directory-based system

• Key properties:
Conclusion

- Proposed **ScalableBulk**: protocol for bottleneck-free commit of chunks in lazy directory-based system

- Key properties:
  - Enables multiple concurrent chunk commits that use the same directory module
Conclusion

• Proposed **ScalableBulk**: protocol for bottleneck-free commit of chunks in lazy directory-based system

• Key properties:
  • Enables multiple concurrent chunk commits that use the same directory module
  • Thanks to use of signatures
Conclusion

- Proposed **ScalableBulk**: protocol for bottleneck-free commit of chunks in lazy directory-based system

- Key properties:
  - Enables multiple concurrent chunk commits that use the same directory module
  - Thanks to use of signatures
  - Eliminates all centralized structures and global communications
Conclusion

- Proposed **ScalableBulk**: protocol for bottleneck-free commit of chunks in lazy directory-based system

- Key properties:
  - Enables multiple concurrent chunk commits that use the same directory module
    - Thanks to use of signatures
  - Eliminates all centralized structures and global communications

- Results: practically eliminates all commit stall overhead for 64 processors
Conclusion

• Proposed **ScalableBulk**: protocol for bottleneck-free commit of chunks in lazy directory-based system

• Key properties:
  • Enables multiple concurrent chunk commits that use the same directory module
    • Thanks to use of signatures
  • Eliminates all centralized structures and global communications

• Results: practically eliminates all commit stall overhead for 64 processors
• Effectively enables a large-scale chunk-based machine
# Message Order Analysis

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Successful Commit</th>
<th>Failed Commit: Collision Module is the Leader</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader</td>
<td>R:commit.request → S:g → R:g → (S:commit.success &amp; S:g.success (multicast) &amp; S:bulk.inv (multicast)) → R:bulk.inv.ack (several) → S:commit.done (multicast)</td>
<td>R:commit.request → (S:g.failure (multicast) &amp; S:commit.failure). Discard any later commit_recall. or R:commit.request → R:commit.request → (S:g.failure (multicast) &amp; S:commit.failure)</td>
</tr>
<tr>
<td>Non-Leader</td>
<td>(R:commit.request &amp; R:g) → S:g → R:g.success → R:commit.done</td>
<td>R:commit.request &amp; R:g.failure (from leader)</td>
</tr>
</tbody>
</table>

**Table 4.** Messages sent and received by a directory module.

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Failed Commit: Collision Module is not the Leader</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader</td>
<td>R:commit.request → S:g → R:g.failure → S:commit.failure</td>
</tr>
<tr>
<td>Before Collision Module</td>
<td>(R:commit.request &amp; R:g) → S:g → R:g.failure (from Collision Module)</td>
</tr>
<tr>
<td>Collision Module</td>
<td>(R:commit.request &amp; R:g) → S:g.failure (multicast). Discard any later commit_recall. or</td>
</tr>
<tr>
<td></td>
<td>(R:commit.request &amp; R:commit.request) → R:g → S:g.failure (multicast) or</td>
</tr>
<tr>
<td></td>
<td>(R:g &amp; R:commit.request) → R:commit.request → S:g.failure (multicast)</td>
</tr>
<tr>
<td>After Collision Module</td>
<td>R:commit.request &amp; R:g.failure</td>
</tr>
</tbody>
</table>
ScalableBulk: Scalable Cache Coherence for Atomic Blocks in a Lazy Environment

Xuehai Qian, Wonsun Ahn, Josep Torrellas
University of Illinois

http://iacoma.cs.uiuc.edu/