The Impact of RDMA on Agreement

Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra Marathe, Igor Zablotchi
Data Center Technology: RDMA

Remote Direct Memory Access (RDMA)

RDMA: No involvement of host CPU!
Data Center Technology: RDMA

- Can choose RDMA connections and permissions
- Can give different permissions for different memory regions

Message and Memory (M&M) model [ABC-GPT’18]

<table>
<thead>
<tr>
<th>Process failure</th>
<th>R1</th>
<th>R1 &amp; R2</th>
<th>R2</th>
<th>—</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1: read</td>
<td>R1</td>
<td>R1 &amp; R2</td>
<td>R2</td>
<td>—</td>
</tr>
<tr>
<td>p3: write</td>
<td></td>
<td></td>
<td>R2</td>
<td>—</td>
</tr>
<tr>
<td>p6: read &amp; write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2, p5: none</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Representing an M&M Network

Only represent memory connections
Permissioned M&M Model

- Decouple processes and memory
- changePermission function can be called on memories
- Failures can occur on processes or memory
  - Byzantine failures of processes
- When memory fails, all regions fail together

Protocol specifies response to permission requests
Main Take-Away

- **Byzantine** or **crash** failures of processes

RDMA improves tradeoff between fault tolerance and performance for agreement

- **Common-case** running time

Common case:
- Synchronous
- No Failures

Running time (agreement):
- Time until *first process* decides

Best case performance
Worst case resilience
Agreement: Previous Results

**Agreement**
- All processes get input, must output the same value

Crash-only (consensus): Processes might **crash**: never respond
Byzantine: Processes might be **Byzantine**: malicious

**Crash Validity**
- If a process outputs value v, some process got v as input

**Weak Byzantine Validity**
- If there are no faulty processes, if a process outputs value v, some process got v as input
## Agreement: Known Results

<table>
<thead>
<tr>
<th></th>
<th>Crash</th>
<th>Byzantine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tolerance</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Common Case Performance</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Message Passing</strong></td>
<td>$n &gt; 2f$</td>
<td>2 [Lamport’98]</td>
</tr>
<tr>
<td><strong>Shared Memory</strong></td>
<td>$n &gt; f$</td>
<td>4 [GL’02]</td>
</tr>
<tr>
<td><strong>RDMA (this paper)</strong></td>
<td>$n &gt; f$</td>
<td>2</td>
</tr>
</tbody>
</table>

* Depends on primitives used. No results for weak Byzantine agreement with single-writer multi-reader registers
Outline

RDMA improves tradeoff between fault tolerance and performance

- All results handle $f_m < m/2$ memory failures
- **Crash-only Consensus:**
  - Tolerance: $n > f$
  - Common-case running time: 2 message delays
- **Weak Byzantine Agreement:**
  - Tolerance: $n > 2f$
  - Common-case running time: 2 message delays
- Impossible without messages and memory

See paper
Handling Memory Failures

Replication: Treat all memories the same
Send all write/read requests to all memories, wait to hear acknowledgement from majority

Instead of many faulty memories, we can now think of one non-faulty memory!
Outline

RDMA improves tradeoff between fault tolerance and performance

- All results handle \( f_m < \frac{m}{2} \) memory failures
- **Crash-only Consensus:**
  - Tolerance: \( n > f \)
  - Common-case running time: 2 message delays
- **Weak Byzantine Agreement:**
  - Tolerance: \( n > 2f \)
  - Common-case running time: 2 message delays
- Impossible without messages and memory
Disk Paxos

Idea: run classic message passing algorithm, but replace sends and receives with reads and writes

To send: write your message in your slot in all disks; wait for majority to respond
To receive: read others’ slots in all disks; wait for majority to respond

What happens in common-case execution?
• p1 proposes by writing to everyone
• p1 must read all slots to ensure it’s the only proposer

P1 doesn’t know it’s a good execution!

Time (in message delays):
2
2
total: 4
In Disk Paxos, proposer must read every value from every disk to know whether someone is competing with it.

Idea: leverage RDMA dynamic permissions to get rid of this step.

If a proposer finished writing without losing permission, there is no one competing with it.

4 message delays --> 2 message delays

I've lost my permission

I will give write permission only to the last person who requested it.

Request permission from m3
Outline

RDMA improves tradeoff between fault tolerance and performance

- All results handle $f_m < m/2$ memory failures
- **Crash-only Consensus:**
  - Tolerance: $n > f$
  - Common-case running time: 2 message delays
- **Weak Byzantine Agreement:**
  - Tolerance: $n > 2f$
  - Common-case running time: 2 message delays
- Impossible without messages and memory

See paper
Byzantine Algorithm
Breakdown

Three pieces:

• **CheapQuorum**: Fast (2-message delay) algorithm that aborts at first sign of trouble
  • Use permissions to get speed

• **Robust Backup**: Slow algorithm that is tolerant to \( n > 2f \) Byzantine failures
  • Shared memory algorithm

• **Preferential Paxos**: Glue

New shared memory result
Robust Algorithm

- High level idea: run Paxos, but replace messaging primitives (send/receive) with special **non-equivocating broadcast/deliver**
  
  Equivocation:

  - If we can prevent equivocation, then we can solve weak Byzantine agreement with $n > 2f$  
    [CJKR’12]
General Scheme

Each process gets its own SWMR region

Protocol: Sign and copy over everything that you see

Can now verify that others read the same value
Summary

**RDMA improves tradeoff between fault tolerance and performance**

- **Crash Consensus algorithm:**
  - $n > f$ fault tolerance, 2-message delays

- **Byzantine Agreement algorithm:**
  - $n > 2f$ fault tolerance, 2-message delays

- Impossible in a single algorithm in shared memory or message passing alone

*RDMA is more powerful, opens up many opportunities for better algorithms!*

Thank you!