“MINOS*: Distributed Consistency and Persistency Protocol Implementation & Offloading to SmartNICs”

Antonis Psistakis, Fabien Chaix*, and Josep Torrellas
University of Illinois Urbana-Champaign, USA
*FORTH, Greece
{psistaki, torrela}@illinois.edu, chaix@ics.forth.gr

*MINOS: King of Crete island (Greek mythology)
Introduction

• Datacenters focus
  • Performance (latency and tput)
  • Availability
  • Reliability

To achieve the above:
• Replicate data across Distributed Datastores
• Persistency
Introduction

*Leaderless* distributed systems: All participating nodes can process any client request

👍 Performance
👎 Programming complexity, Recovery
Background: Definitions

- **Consistency Model**: When updates become visible (replicated) to all nodes
  - **Linearizable** (Lin): A client write must update all the replica nodes in the system before it completes.

- **Persistency Model**: When updates are persisted to non-volatile memory (NVM)
  
<table>
<thead>
<tr>
<th>Persistency Model*</th>
<th>When is an update persisted in a node’s NVM?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous (Synch)</td>
<td>When the local volatile memory is updated</td>
</tr>
<tr>
<td>Read-Enforced (REnf)</td>
<td>Before the value updated is read</td>
</tr>
<tr>
<td>Scope</td>
<td>At the end of the scope</td>
</tr>
<tr>
<td>Eventual (Event)</td>
<td>Sometime in the future</td>
</tr>
</tbody>
</table>

* Kokolis et al., “Distributed Data Persistency”, MICRO 2021
Background: Definitions

- **Type of Nodes**
  - **Coordinator**: The node that receives the client request
  - **Follower**: The nodes with replicas. Need to participate in the update

- **Type of Messages**

<table>
<thead>
<tr>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidation (INV)</td>
</tr>
<tr>
<td>Acknowledgement (ACK)</td>
</tr>
<tr>
<td>Validation (VAL)</td>
</tr>
</tbody>
</table>

Background: DDP <Lin, Synch> Example

Coordinator

WR

Update Mem

INV (+data)

WR latency

ACK

END

VAL

Follower

INV (+data)

Update Mem

RD

RD Stall

RD Return

ACK

VAL
Contributions

1. Novel algorithms for real-system implementation of a Leaderless system, supporting various consistency and persistency models (MINOS-B)

2. New architecture that offloads the models into SmartNICs (MINOS-O)
Contribution: MINOS-Baseline (MINOS-B)

- Set of novel leaderless algorithms that efficiently implement consistency and persistency (DDP) models
- MINOS-B relies on three elements to support concurrent and conflicting writes:
  A. Logical Timestamps & Obsolete Writes
  B. Lock Types
  C. Lock Ownership
MINOS-B: Logical Timestamps & Obsolete Writes

**Logical Timestamps**
- Used to maintain order of requests
- Each write operation carries its own timestamp

**Obsolete Writes**
- Write that reaches a node and the record is already updated by a later update
- Returns immediately to the sender without updating record
MINOS-B: Lock Types

Naïve approach: Use plain Locks
MINOS-B: Lock Types

What really needs lock protection?

1. Prevent reads on records currently being written
   • Scope: Client Write
2. Prevent concurrent writes on the same record
   • Scope: Local Write

* Note: Persist is done atomically in a REDO log, okay to be re-ordered
MINOS-B: Lock Types

Our Approach

1. Prevent reads on records currently being written
   • Scope: Client Write
2. Prevent concurrent writes on the same record
   • Scope: Local Write

Optimization: Snatching Read-Lock ownership
MINOS-B: Read-Lock Ownership Example

Node 1 Coordinator
- Key: 0
- TS: 0
- WR1

Node 2 Coordinator
- Key: 0
- TS: 1
- WR2

Node 3 Follower
- INV1
- INV2
- VAL1
- VAL2
- WR1: RDLOCK
- WR2: Snatch RDLOCK
- WR2: WRLOCK
- WR2: Persist
- WR1: RDUNLOCK
- WR2: RDUNLOCK

Table:

<table>
<thead>
<tr>
<th>NODE 3</th>
<th>KEY 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDLOCK</td>
<td>&lt;WR2&gt;</td>
</tr>
<tr>
<td>WRLOCK</td>
<td>0</td>
</tr>
<tr>
<td>Volatile State</td>
<td>&lt;WR2&gt;</td>
</tr>
</tbody>
</table>
MINOS-B: Full <Lin, Synch> Algorithm

1. Process new WR for key k
   2. If Obsolete(TS_WR):
      3. Exit
      4. Snatch RDLock(k)
         // RDLock is set.
         // This thread may be the RDLock Owner.
      5. WRLock(k)
         // This thread can perform a local-write.
   6. If !Obsolete(TS_WR):
      7. Send INVs
      8. Update Mem
      9. WRUnlock(k)
   10. If INV sent:
      11. Persist to NVM
      12. Spin for all ACKs
      13. If RDLock_Owner(k) == Me:
         14. RDUnlock(k)
      15. If INV sent: Send VALs

16. Process new INV for key k
17. If Obsolete(TS_WR):
18.   Send ACK
19. Exit
20. Snatch RDLock(k)
21. WRLock(k)
22. If !Obsolete(TS_WR):
23.   Update Mem
24. WRUnlock(k)
25. If updated Mem: Persist to NVM
26. Send ACK
27. Process new VAL for key k
28. If RDLock_Owner(k) == Me:
29.   RDUnlock(k)
30. < Not applicable >
Sources of Overhead

- Same msg sent to multiple receivers one-by-one
- Heavy involvement of the Follower CPUs
- Expensive PCIe crossings
- Software overhead of concurrency control

MINOS-B <Lin, Synch> Algorithm
Contribution: MINOS-Offload (MINOS-O)

**Idea:** Offload consistency and persistency operations to SmartNICs

SmartNIC enhancements:

1. Equipped with both *volatile*- and *non-volatile* memories
2. Metadata is *cache-coherent* with host CPU
3. Batching of messages between Host ⇔ SmartNIC
4. Broadcast support
MINOS-O Features

MINOS-B <Lin, Synch> Algorithm

MINOS-O <Lin, Synch> Algorithm

4. Metadata Coherence
Methodology

Simulated System
• 2-16 nodes, 5 cores each (2.1 GHz)
• SmartNIC with 8 cores at 2GHz

Benchmarks
• Microbenchmark: YCSB
• Macrobenchmark: DeathStar Benchmark*

* Gan et al., “An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems”, ASPLOS19
MINOS-O reduces read latency by 3.1x on average
Conclusion

- **MINOS-Baseline (MINOS-B):** set of new algorithms for efficient implementation of leaderless consistency and persistency models

- **MINOS-Offload (MINOS-O):** offloads MINOS-B algorithms to SmartNICs
  - 2.7x average latency reduction over MINOS-B
  - 2.4x average throughput increase over MINOS-B
  - End-to-end microservice average latency: 35% reduction over MINOS-B
30th IEEE International Symposium on High-Performance Computer Architecture (HPCA 2024)

“MINOS: Distributed Consistency and Persistency Protocol Implementation & Offloading to SmartNICs”

Antonis Psistakis, Fabien Chaix*, and Josep Torrellas
University of Illinois Urbana-Champaign, USA
FORTH, Greece
{psistaki, torrela}@illinois.edu, chaix@ics.forth.gr

Thank you!

Scan to read our paper

Edinburgh, Scotland
March 5th, 2024
Backup Slides
MINOS-B: Changes for other DDP Models

**Coordinator**
- Spin for all ACK-Cs
- If INVs sent:
  - Send VAL_Cs
  - Spin for all ACK_Ps
  - Send VAL_Ps

(i) <Lin, Strict>

**Coordinator**
- Persist to NVM in background
- Spin for all ACK-Cs
- Return to Client
- Receive all ACK_Ps

(ii) <Lin, Strict>

**Coordinator**
- Send [INV]_{sc}
- If [INV]_{sc} sent:
  - Send [ACK_C]_{sc}, [ACK_P]_{sc}, and [VAL_C]_{sc}

(a) <Lin, Event>

**Coordinator**
- Process new [INV]_{sc} for key k
- Send [PERSIST]_{sc}
- Spin for all [ACK_P]_{sc}
- Send [VAL_P]_{sc}

(b) <Lin, Event>

**Coordinator**
- Receive [PERSIST]_{sc}
- When persists in scope sc are done:
  - Send [ACK_P]_{sc}
  - Process [VAL_P]_{sc}

(c) <Lin, Event>

**Coordinator**
- Send [INV]_{sc}
- If [INV]_{sc} sent:
  - Send [ACK_C]_{sc}, [ACK_P]_{sc}, and [VAL_C]_{sc}

(ii) <Lin, Event>

**Coordinator**
- If INVs sent: Send VAL_Cs
- Spin for all ACK_Cs
- Send VAL_Ps

(v) <Lin, Event>

**Coordinator**
- Send ACK_C
- Send ACK_P
- Update Mem
- Send ACK_C
- Send ACK_P
- Process new VAL_C for key k
- Process new VAL_P for key k

(g) <Lin, REnv>

**Coordinator**
- Persist to NVM in background
- Spin for all ACK_Cs
- Return to Client
- Receive all ACK_Ps

(iii) <Lin, REnv>

**Coordinator**
- Send ACK_C
- Send ACK_P
- Update Mem
- Send ACK_C
- Send ACK_P
- Process new VAL_C for key k

(iv) <Lin, REnv>

**Coordinator**
- Persist to NVM in background
- Spin for all ACK_Cs
- Return to Client
- Receive all ACK_Ps

(ii) <Lin, REnv>

**Coordinator**
- Send ACK_C
- Send ACK_P
- Update Mem
- Send ACK_C
- Send ACK_P
- Process new VAL_C for key k

(vi) <Lin, REnv>

**Coordinator**
- Persist to NVM in background
- Spin for all ACK_Cs
- Return to Client
- Receive all ACK_Ps

(vii) <Lin, Scope>

**Coordinator**
- Send ACK_P
- Update Mem
- Send ACK_P
- Process new VAL_P for key k

(j) <Lin, Scope>
MINOS-B: Write Latency Breakdown

- Stricter persistency models persist in the critical path of a write → higher overheads
- High communication overhead across all pers. models (51-73%)
MINOS-O reduces read latency by 3.1x on average
MINOS-O reduces average end-to-end latency by 35% (max: 39%)
Methodology

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes</td>
<td>5</td>
</tr>
<tr>
<td>CPU per node</td>
<td>Xeon E5-2450 (5 cores, 2.1 GHz)</td>
</tr>
<tr>
<td>Main memory per node</td>
<td>16GB of DRAM (DDR3-1600)</td>
</tr>
<tr>
<td>NIC per node</td>
<td>Mellanox MX354A FDR CX3</td>
</tr>
<tr>
<td>Emulated NVM per node</td>
<td>1295 ns to persist 1KB of data</td>
</tr>
</tbody>
</table>

Distributed machine running MINOS-B.

For the communication of nodes we ported and extended the eRPC* library.

<table>
<thead>
<tr>
<th>Node</th>
<th>Number of Nodes</th>
<th>Host</th>
<th>SmartNIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Cores</td>
<td>2, 4, 5 (default), 6, 8, 10, or 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core frequency</td>
<td>5</td>
<td>2.1 GHz</td>
<td>8</td>
</tr>
<tr>
<td>Synchronization latency</td>
<td>42 ns</td>
<td>105 ns</td>
<td></td>
</tr>
</tbody>
</table>

**Communication Link**

- PCIe between Host and SmartNIC: 500 ns [35] 6.25 GB/s [43]
- Network link between SmartNICs: 150 ns 7 GB/s [39]

**MINOS-O Parameters**

- vFIFO & dFIFO latency (wr 1KB): 465 ns and 1295 ns
- vFIFO & dFIFO size: 5 and 5 entries
- Send one INV and send one ACK: 200 ns and 100 ns
- Time between consecutive msgs: 100 ns (with no broadcast support)

Parameters of the simulated system.

* Kalia et al., “Datacenter RPCs can be General and Fast.”, NSDI 2019
Rest of MINOS-O algorithms can be found in our paper