Tailwind: Fast and Atomic RDMA-based Replication

Yacine Taleb, Ryan Stutsman, Gabriel Antoniu, Toni Cortes
In-Memory Key-Value Stores

- General purpose in-memory key-value stores are widely used nowadays
In-Memory Key-Value Stores

- General purpose in-memory key-value stores are widely used nowadays

- Recent systems can process millions of requests/second/machine: E.g. RAMCloud, FaRM, MICA, ...
In-Memory Key-Value Stores

• General purpose in-memory key-value stores are widely used nowadays

• Recent systems can process millions of requests/second/machine: E.g. RAMCloud, FaRM, MICA, ...

• Key enablers: eliminating network overheads (e.g., kernel bypass) and leveraging multicore architectures
In-Memory Key-Value Stores

- CPU is becoming a bottleneck in modern kv-stores
  - Random memory access, key-value GET/PUT processing
In-Memory Key-Value Stores

- CPU is becoming a bottleneck in modern kv-stores
  - Random memory access, key-value GET/PUT processing

- Persistent in-memory kv-stores have to replicate data to survive failures
In-Memory Key-Value Stores

- CPU is becoming a bottleneck in modern kv-stores
  -> Random memory access, key-value GET/PUT processing

- Persistent in-memory kv-stores have to replicate data to survive failures
In-Memory Key-Value Stores

• CPU is becoming a bottleneck in modern kv-stores
  -> Random memory access, key-value GET/PUT processing

• Persistent in-memory kv-stores have to replicate data to survive failures
  -> Replication contends with normal request processing
In-Memory Key-Value Stores
In-Memory Key-Value Stores

Client

PUT(Y)

CPU

DRAM

Backup copies

Primary copies
In-Memory Key-Value Stores

Replicate X
Primary

CPU

Client
PUT(Y)

Backup copies
Primary copies

DRAM

Y
In-Memory Key-Value Stores

Primary

Replicate X

CPU

Backup copies

Primary copies

DRAM

PUT(Y)

Client
In-Memory Key-Value Stores

- Replicate X
- Replicate X'
- Replicate X''
- PUT Y
- GET Y
- PUT Y'

Primary copies
Backup copies
Primary copies
Replication in In-Memory Key-Value Stores

- Replication impedes modern kv-stores
Replication in In-Memory Key-Value Stores

- Replication impedes modern kv-stores and collocated applications
Replication in In-Memory Key-Value Stores

- How to mitigate replication overhead?
Replication in In-Memory Key-Value Stores

- How to mitigate replication overhead?
- Techniques like Remote Direct Memory Access (RDMA) seem promising
Replication in In-Memory Key-Value Stores

- Replicate X
- Replicate X'
- Replicate X''

Primary copies
Backup copies

Primary

NIC
CPU

PUT Y
GET Y
PUT Y'
RDMA-Based Replication

Primary copies

Backup copies

Primary

Replicate X

Replicate X’

Replicate X’’

NIC

CPU

DMA

DRAM

Client

GET Y

PUT Y

PUT Y’
RDMA-Based Replication

- Primary
- Replicate X
- Replicate X'
- Replicate X''

Backup copies: X, X', X''

Primary copies: Y, Y'

NIC

DMA

CPU
Existing Protocols

- Many systems use one-sided RDMA for replication

FaRM NSDI’14, DARE HPDC’15, HydraDB SC’16

Still involves backup’s CPU, defeats RDMA purpose!
Outline

• Context

• Existing RDMA-based replication protocols

• Tailwind’s design

• Evaluation

• Conclusion
RDMA-based Replication

- Why not just perform RDMA and leave target idle?
RDMA-based Replication

- Why not just perform RDMA and leave target idle?
RDMA-based Replication

- Why not just perform RDMA and leave target idle?
RDMA-based Replication

- Why not just perform RDMA and leave target idle?
RDMA-based Replication

- Why not just perform RDMA and leave target idle?

The backup cannot “guess” the length of data, it needs some metadata.
RDMA-based Replication Metadata

- A backup needs to insure the integrity of log-backup data + length of data
RDMA-based Replication Metadata

- The last checksum can cover all “previous” log entries
RDMA-based Replication Metadata

- The last checksum can cover all “previous” log entries
RDMA-based Replication
Second Try

- The primary writes metadata on a remote buffer
RDMA-based Replication
Second Try

- The primary writes metadata on a remote buffer

Current RDMA implementations can only target a single, contiguous memory region
RDMA-based Replication
Second Try

- The primary writes metadata on a remote buffer

2x Messages, RPCs would be more efficient
RDMA-based Replication
Second Try

- The primary writes metadata on a remote buffer
The primary writes metadata on a remote buffer
RDMA-based Replication
Second Try

- The primary fails in the midst of updating metadata!
RDMA-based Replication
Second Try

- Backup data unusable!
RDMA-based Replication
Third Try

- The primary replicates A and corresponding metadata with a single RDMA
RDMA-based Replication
Third Try

- The primary replicates B and corresponding metadata, right after A
RDMA-based Replication
Third Try

- The primary fully replicates C, but partially the metadata

Corrupt metadata invalidates all backup log
RDMA-based Replication
Fourth Try

- The primary replicates A and corresponding metadata
RDMA-based Replication
Fourth Try

- The primary partially replicates B, then fails
RDMA-based Replication
Third Try

- The backup checks if objects were fully received
RDMA-based Replication
Third Try

- The backup checks if objects were fully received

Only fully-received and correct objects are recovered!
Tailwind

- Keep the same client-facing interface (RPCs)
- Strongly-consistent primary-backup systems
- Appends only a 4-byte **CRC32 checksum** after each record
- Relies on **Reliable-Connected** queue pairs: messages are delivered at most once, in order, and without corruption
- Stop failures
RDMA Buffers Allocation

- A primary chooses a backup, and requests an RDMA buffer
RDMA Buffers Allocation

- A primary chooses a backup, and requests an RDMA buffer
RDMA Buffers Allocation

- All subsequent replication requests are performed with one-sided RDMA WRITEs
RDMA Buffers Allocation

- When the primary fills a buffer, it will choose a backup and repeat all steps

Buffers are asynchronously flushed to secondary storage, then they can be reused.
Tailwind: Failures

- Failures can happen at any moment
- RDMA complicates primary replica failures
- Secondary replica failures are naturally dealt with in storage systems
Failure scenarios: Fully Replicated Objects

- Case 1: The object + its metadata are correctly transferred
Failure scenarios: Fully Replicated Objects

- Case 1: The object + its metadata are correctly transferred
Failure scenarios: Fully Replicated Objects

- Case 1: The object + its metadata are correctly transferred
Failure scenarios: Fully Replicated Objects

- Case 1: The object + its metadata are correctly transferred

The last object must always be a checksum + Checksums are not allowed to be zeroed
Failure scenarios: Partially Written Checksum

- Case 2: Partially transferred checksum
Failure scenarios: Partially Written Checksum

- Case 2: Partially transferred checksum
Failure scenarios: Partially Written Checksum

- Case 2: Partially transferred checksum

Backups re-compute checksum during recovery and compare it with the stored one
Failure scenarios: Partially Written Object

- Case 3: Partially transferred object

Metadata act as an end-of-transmission marker
Outline

• Context

• Existing RDMA-based replication protocols

• Tailwind’s design

• Evaluation

• Conclusion
Implementation

- Implemented on the RAMCloud in-memory kv-store
- Low latency, large scale, strongly consistent
- RPCs leverage fast networking and kernel bypass
- Keeps all data in memory, durable storage for backups
RAMCloud Threading Architecture

1. **PUT(B)**: Data is put into the non-volatile buffer.
2. **Replicate(B)**: Data is replicated to other nodes.
3. **Replicate(C)**: Data is replicated to other nodes.
4. **Replicate(A)**: Data is replicated to other nodes.

- **Primary DRAM storage**: Stores primary data.
- **Worker Core**: Processes data.
- **Non-volatile Buffer**: Stores data persistently.
- **Client**: Initiates PUT and GET operations.
Evaluation Configuration

- Yahoo! Cloud Serving Benchmark (Workloads A (50% PUT), B (5% PUT), WRITE-ONLY)
- 20 million - 100 bytes objects + 30 byte/key
- Requests generated with a Zipfian distribution
- RAMCloud replication vs Tailwind (3-way replication)

<table>
<thead>
<tr>
<th>CPU</th>
<th>Xeon E5-2450 2.1 GHz 8 cores, 16 hw threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>16 GB 1600 MHz DDR3</td>
</tr>
<tr>
<td>NIC</td>
<td>Mellanox MX354A CX3 @ 56 Gbps</td>
</tr>
<tr>
<td>Switch</td>
<td>36 port Mellanox SX6036G</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 15.04, Linux 3.19.0-16, MLX4 3.4.0, libibverbs 1.2.1</td>
</tr>
</tbody>
</table>
Evaluation Goals

- How much CPU cycles can Tailwind save?
- Does it improve performance?
- Is there any overhead?
Evaluation: CPU Savings

Values are aggregated over a 4-node cluster WRITE-ONLY workload

CPU cores utilization (dispatch)

CPU cores utilization (worker)
Evaluation: Latency Improvement

Throughput (Kops)

Median Latency (µs)

Tailwind
RAMCloud

Durable writes take 16µs under heavy load

99th Percentile Latency (µs)

Throughput (Kops)

2x

3x
Evaluation: Throughput

Even with 5% of PUTs, Tailwind increase throughput by 30%
Evaluation: Recovery Overhead

Recoveries with up to 10 million objects
Related Work

- One-sided RDMA systems: Pilaf ATC’13, HERD SIGCOMM’14, FaRM NSDI’14, DrTM SOSP’15, DrTM + R Eurosys’16, ...

- Mitigating replication overheads/Tuning consistency: RedBlue OSDI’12, Correctables OSDI’16

- Tailwind reduces replication CPU footprint and improves performance without sacrificing durability, availability, or consistency
Conclusion

- Tailwind leverages one-sided RDMA to perform replication and leaves backups completely idle
- Provides backups with a protocol to protect against failure scenarios
- Reduces replication induced CPU usage while improving performance and latency
- Tailwind preserves client-facing RPC

Thank you! Questions?