CliqueMap: Productionizing an RMA-Based Distributed Caching System

Arjun Singhvi, Aditya Akella, Maggie Anderson, Rob Cauble, Harshad Deshmukh, Dan Gibson, Milo M. K. Martin, Amanda Strominger, Thomas F. Wenisch, Amin Vahdat
Introduction / Summary

In-memory key-value caching/serving systems are crucial building blocks of user-facing services throughout the industry (Twemcache\textsuperscript{(osdi20)}, CacheLib\textsuperscript{(osdi20)}, …)

Remote Memory Access (RMA):
- Benefits: Performance/efficiency benefits
- Downsides: Limited programmability/narrow primitives
- Production Challenges
  - Delivering high availability and low cost
  - Balancing CPU- and RAM-efficiency
  - Evolving the system over time
  - Multi-language serving ecosystems
  - Navigating heterogeneous datacenters

How do we productionize an RMA-based distributed caching system?
Hybrid RMA+RPC caching system in production use at Google 3+ years.
- Serves >1PB DRAM, >150M QPS
- RMAs on the critical serving path
- RPCs for mutations & other functions
- Simple “2xR” lookup protocol amenable to different underlying RMA technologies (RDMA, PonyExpress, 1RMA)

A 2xR-style R=1 Lookup operation using RMA primitives. A first operation to a predictable location finds the datum in an index. A second, dependent operation retrieves the datum.
**RPC or RMA? False dichotomy.**

**RMAs** *[No application code runs on target]* offer narrow but efficient primitives.

**RPCs** *[Wherein arbitrary application code runs/responds on target]* offer easier productionization and high flexibility.

Hybrids like CliqueMap leverage the strengths of both: RMA for most/important operations to gain efficiency, RPC when programmability is needed.
CliqueMap Approach and Building Blocks

**Self-verification**: A lookup self-verifies its outcome by strongly checksumming data, key, and metadata.

**Retry at the Right Layer of the Stack**: E.g., checksum failures repeat the lookup. Metadata inconsistencies (e.g., during a rollout) reload configuration.
Challenge: Availability/Cost Tradeoffs

**Tension with RMA:** Synchronizing RMAs, tolerating failures.

**CliqueMap’s Approach:**
- Modes for R=1, R=2, R=3.2 for tuning availability/cost tradeoffs
- RPCs for mutations; RMAs are self-verifying
- Data migration for maintenance events
- Tunable on demand repair

A 2xR-style R=3.2 Quorumed Lookup operation. By establishing a quorum (majority vote) on metadata, a slow, absent, or inconsistent replica can be tolerated.
Challenge: Memory & CPU Efficiency

**Tension with RMA**: Memory registration is expensive/subtle; needs to be done off the critical path.

**CliqueMap’s Approach**: Dynamic Backend Scaling
- Start expanding memory when usage above watermark (RPC-triggered)
- Clients can discover new backend geometries lazily, refresh metadata

Plot of memory usage over time after Dynamic Backend Scaling’s initial rollout. Initially, capacity was simply slightly overprovisioned - this memory could be released. At ~Week 8, demand on corpus fell and more memory could be safely refunded.
Challenge: Evolution over Time

*Tension with RMA*: RMA exposes in-memory binary formats, making iteration difficult.

*CliqueMap’s Approach*: Metadata verification during checksumming enables protocol versioning. Entirely new primitives can be introduced.

SCAR was a major feature introduction that occurred post-productionization; evolution-friendly retry-based design enabled a transition wherein the logical 2xR lookup strategy could be flattened to a single round-trip, leading to efficiency improvements across all layers of infrastructure.
Challenge: Language Interoperability

_Tension with RMA:_ C/C++ predominance

_CliqueMap’s Approach:_

- Launch a subprocess containing the normal C++ CliqueMap libraries
  - IPC solutions per target language
    - Go, Python → Named Pipes
    - Java → Shared Memory
- Enables established, large-scale infrastructure with substantial non-C++ components to adopt CliqueMap.
Challenge: Hardware heterogeneity

*Tension with RMA:* Wire Interoperability, performance expectations, mixed-age hardware

**CliqueMap’s Approach:**

- Resilient, generic high-level protocols (2xR) suitable to different underlying RMA implementations (e.g., SCAR)
- Evolve over time, embrasure of programmable NICs
Coming up

A Deeper look at R=3.2

- Backend Memory Layout in Detail
- 2xR GET/SET Example
- Enduring Failures
CliqueMap Backend Memory Layout

Backend hashtable layout chosen to be amenable to self-verification, retries, and evolution.

- Backend can relocate DataEntires, e.g., to defrag
- Checksum covers index and data end-to-end (client can detect inconsistencies and retry)
- Fields include enough metadata to hint at the right kind of retry
R=3.2: Quoruming and Versioning

SET K=V₁ @7.1

Bucket Fetches K
R=3.2: Quoruming and Versioning

SET $K=V_1$ @7.1

Bucket Fetches $K$

$C_1$ Quorum $V_0$

Quorum $V_0$
R=3.2: Quoruming and Versioning

SET K=V1 @7.1

C1 SET Complete

Quorum V0

V0 → V1

Quorum V1

V0 → V1

Bucket Fetches K

C1 Quorum V0

Quorum V0

C1 Quorum V0

V0 → V1

V0 → V1

V0 → V1
R=3.2: Quoruming and Versioning

SET K=V₁ @7.1

C₁ Quorum V₀

Data Fetch

C₂ GET Complete, Checksum Failure

Bucket Fetches K
R=3.2 with Unplanned Failures

R=3.2 with repair preserves performance across single unplanned failures.
R=3.2 with Planned Maintenance/Upgrades

R=3.2 with warm sparing maintains a clean quorum during planned maintenance events.
Closing Remarks

Leverage RPC, in composition with RMA, to maintain post-deployment agility

Enable multi-language software ecosystems

Don’t compromise memory efficiency

Simply design with self-validating server responses and client retries

Programmable NICs offer advantages through specialization

See the paper for many more details!
Thank you!

CliqueMap: Productionizing an RMA-Based Distributed Caching System

Arjun Singhvi, Aditya Akella, Maggie Anderson, Rob Cauble, Harshad Deshmukh, Dan Gibson, Milo M. K. Martin, Amanda Strominger, Thomas F. Wenisch, Amin Vahdat

Google  WISCONSIN  UNIVERSITY OF WISCONSIN-MADISON