Fast key-value stores: An idea whose time has come and gone

Atul Adya, Robert Grandl, Daniel Myers (Google)
Henry Qin (Stanford)
Since we’re in Italy...

“\textit{I come to bury key/value stores, not to praise them.}”
Take-home message

● Remote, in-memory key/value stores are a performance dead-end
● We need to look at end-to-end application performance
● Better performance requires better abstractions
Prelude: What is a key/value store?

- Remote, In-Memory, Key/Value store (RINK)
- Domain-independent API
- Think Memcache or Redis, not Bigtable or HBase
Key/value stores are a thing

- **Academia**: FLOEM (OSDI ‘18), NetCache, KV-Direct (SOSP ‘17), Mega-KV (VLDB ‘15), MemcachedGPU (SoCC ‘15), MemC3 (NSDI ‘13), FaRM, MICA (NSDI ‘14), ...

- **Industry**: Redis / Memcached on all Clouds
  - 44M / 18.7M hits on Google
  - 17.8M for HotOS ;(
How are they used?

Stateless Servers

Client

Application

RINK

Database

Cross-app coordination

Client 1

App 1

RINK

Client 2

App 2
Goals of this talk: #1

Goal: Convince you that key/value stores have outlived their usefulness

- Key/value stores make applications slow
- Industry: please stop using them
- Academia: please stop improving them
Goals of this talk: #2

Goal: Convince you that we can do better

- Idea 1: Better performance by better abstractions
  - Stateful servers or domain-specific in-memory stores
- Idea 2: Build infrastructure to enable Idea 1

Disagree? Find a better solution; we’ll use it.
How can key/value stores be slow?

- NetCache (2017): 2+ billion queries/sec/switch
- KV-Direct (2017): 1.22 billion queries/sec/server
- Mega-KV (2015): 110M queries/sec

All are objectively fast and did interesting work.
End-to-end view of performance

- No developer wants a fast key/value store per se
- Developers want to build fast *applications*
- RINK abstraction pushes costs to applications
  - (Un)marshalling
  - Overreads
  - Network latency
Example: address book service

- Simplified real application ("ProtoCache" in paper)
- Maintains an address book per user
- Imagine implementing using a RINK store

User 1
Name: Bob Jones
Phone: 212-555-1212
Address: 747 6th St...

User 2
Name: Jane Smith
Phone: 718-555-1212
Address: 651 N34th St...
(Un)marshalling

- (Mostly) can’t compute on strings
  - `jsonstr.find("fname: bob")`?
- Need a string ←→ data structure step
- Our experiments: 40% of CPU

User1: 

```
{"fname: 'bob'..."
```

Client

Application

RINK
But wait!

● Is (un)marshalling really fundamental?
  ○ Can’t I just `memcpy(&rink, &myobj)`?

● Yes (it is); no (you can’t)
  ○ Object graphs / pointers
  ○ Cross-language interoperability
  ○ **Software upgrades, schema evolution**
Overreads

- Key/value API forces whole record read
- ProtoCache: 4% of value needed (mean)
- Another system: 7/70 fields, 37% of bytes (mean)
But wait!

- Isn’t this a strawman data model? **No.**
- Non-workable alternatives:
  - Multiple key/value pairs
  - Lists / sets / sparse columns
  - ...
- In general: danger in tying application too closely to “storage” system
Network Latency

- Even with fast networks, large value transfer takes time
- 10MB address book?
  - 80 ms at 1 Gbps
  - 8 ms at 10 Gbps
Remember these?
But wait!

- Isn’t 10MB an absurdly huge value?
- No.

- Research systems often focus on small values
  - Production workloads can have large values
  - Large values exacerbate (un)marshalling, overread, and network latency costs
Industrial vs Research Workloads

![Graph showing value size (KB) for various workloads.](image-url)
Amdahl's Law

- Fixed costs
- (Un)marshalling
- Overreads
- Latency

Academia vs Industry
Our Proposal

- Better abstractions
- New infrastructure
Change the abstraction

- Costs exist regardless of RINK performance
- To reduce / eliminate, change the abstraction
- Store domain-specific application objects, not strings or simple data structures
Original Architectures

Client

Application

RINK

Database

Client 1

App 1

RINK

Client 2

App 2
Revised Architecture: Best Case

- Embed sharded cache directly into application
- One cache access per application operation
- Eliminates (un)marshalling, overreads, network latency
- Relatively common
Revised Architecture: Coordination

- Replace RINK with new server
- Can reduce (un)marshalling, overreads, network latency

Domain-specific RPCs; e.g. ReadContact(userid, email)
Revised Architecture: Fanout

- For non-partitionable workloads, request fanout
- Hybrid of first two models
- Application serves as custom store
Wouldn’t it be nice...

...to have efficient partial reads, RMW?

class Objects<V> {
    // Retrieve object from store.
    V* Get(string key);

    // Return object to store.
    bool Commit(string key, V* value);
};

void HandleAddressLookupRpc(String userId, String contactEmail, Writer out) {
    AddressBook contacts = objects.Get(userId);
    out.write(contacts.lookupByEmail(contactEmail));
    contact.recordAccess();  // Bump hit count.
    objects.Commit(userId, contacts);
}
Why can’t we write code this way?

- Systems are constantly perturbed
- Replication for load, availability
- Fine; let’s make it possible
New Abstraction: LINK Store

- Linked, In-Memory Key/Value Store
- Stores application objects
- Data migration on reconfiguration

```java
class Link<V> {
    interface Marshaller {
        string marshal(V v);
        V unmarshal(string s);
    }
    V* Get(string key);
    bool Commit(string k, V* v);
}
```
Deployment Experience at Google

- Built a LINK prototype with load balancing (Slicer, OSDI 2016) and state migration
- ProtoCache rewritten using a subset of prototype
  - Reduced 99.9% latency by 40% (~750 ms to ~450 ms)
- Events processing system being built
  - No numbers yet, but developers like the abstraction
Summary

- RINK costs are under-appreciated
- Reduce costs by changing architectures
  - Stateful services or domain-specific stores
- LINK to make new architectures easy

Not a LINK fan? Find a better solution; we’ll use it.
Call to the Community

● Please think about end-to-end performance
● Many technical problems to solve, including:
  ○ Replication for load and availability
  ○ Freshness
  ○ Partitioning code between servers and store
● Please help!