On The Composability of the Riak DT Map: Expanding From Embedded To Multi-Key Structures

Christopher Meiklejohn

Basho Technologies, Inc.
Cambridge, MA 02139

cmeiklejohn@basho.com

First Workshop on the
Principles and Practice of Eventual Consistency
April 13, 2014

Overview

Problem statement

- Riak DT provides a composable, convergent replicated dictionary [2]
- Composition is supported through embedding
- Increasing object sizes cause a performance degradation in Riak because of implementation details
- Provide two solutions
  - Provide an alternative composition strategy, composition by reference
  - Provide a partial query mechanism

Customer example

- Social network timelines [6] [5]
  - Manifest objects for each timeline
  - References to each object, stored independently
- Custom merge/prune functions
- Performance degradation
- Lack of causal consistency

Sample Timeline

```
{
 "1397213894":"0beec7"
 "1397213994":"62cdb7"
 "1397214094":"bbe960"
}
```

Riak

- DHT with fixed partition size/count
- Partitions claimed on membership change
- Replication over ring-adjacent partitions (preference lists)
- Sloppy quorums (fallback replicas) for added durability
- Opaque object, single version.

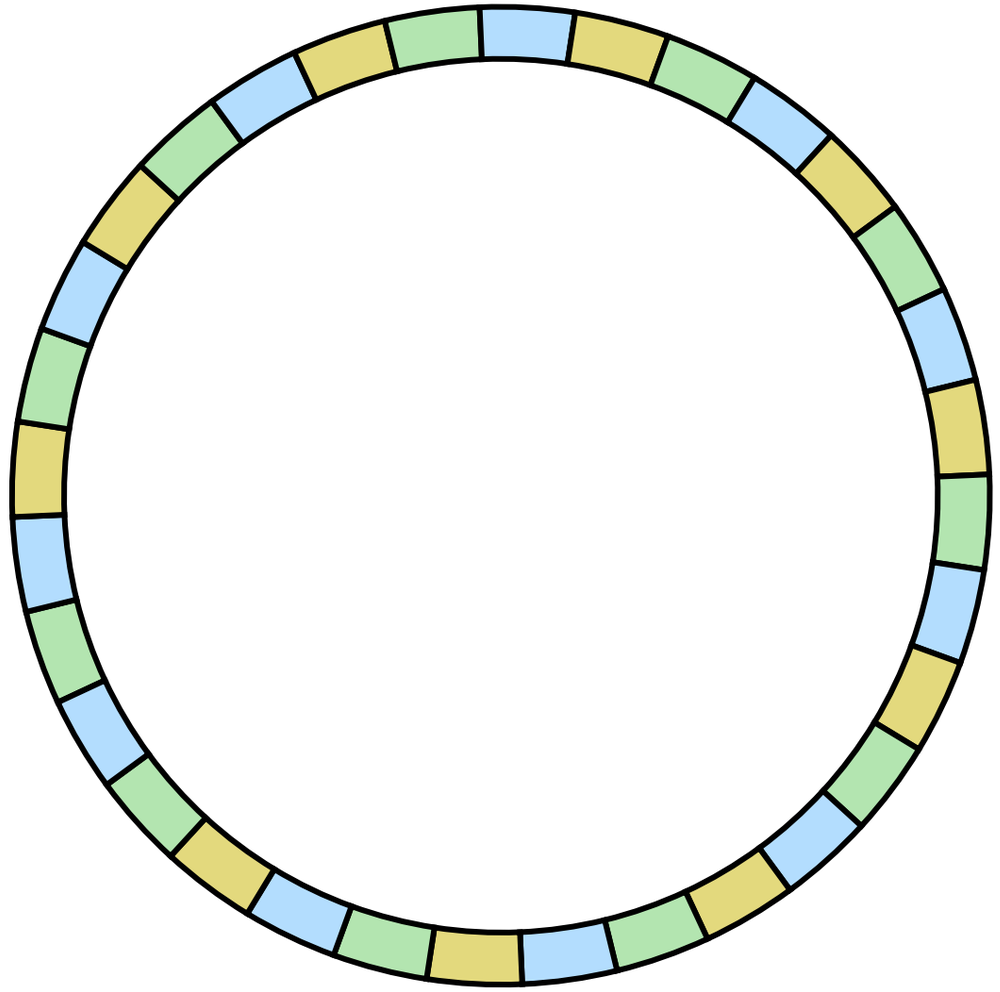

Figure : Ring with 32 partitions and 3 nodes

Distributed Erlang

- Ability to cluster a group of Erlang runtime systems
- Transparent to use when using message passing, links, monitors, etc
- TCP/IP socket based; full mesh network

Problems

- *busy_dist_port* problem [3]
  - distribution channel for outgoing messages fills up
  - pauses sending processes when full
- TCP Incast problem [4]
  - many-to-one communication patterns cause overload
  - switch buffer overload
  - TCP congestion control, TCP slow start

Riak DT Map

- A dictionary
- Field keys are pairs of $(Id, Type)$
- Field values are CvRDTs
- Batched/atomic operations on nested types
- Observed-remove semantic on fields
- Field removals on unseen events are deferred

Riak DT in Riak K/V

- Extends Riak KV's object storage API
- Enables storage of Riak DT CvRDTs in Riak KV
- Exposed as HTTP/PB
- Relies on Riak's bucket types
- Honors Riak's get/put parameters

Map Update via HTTP

```
{
  "update": {
    "goal_counter":  -1,
    "fault_counter": 1,
    "name_register": "Bruins"
  }
}
```

Composition by reference

- Provide a mechanism for composition by reference
  - Bucket type property
  - Generate a unique id for composed CvRDT
    - Name
    - Type
    - Composition level and type
- Use this identifier as the object key for the CvRDT
- Store the CvRDT as a separate object using this key

Read/write coordination

- Write coordination
  - Create a list of all dependent writes which need to happen
  - Fail the entire write if any of the dependent writes fail
  - Update the map object with any new references
- Read coordination
  - Read map object
  - Recursively retrieve references and reassemble map before returning to user
  - Honors quorum parameters provided by Riak

Replica placement of composed objects

- Same primary replica set as map object
  - Decreased parallelization due to serialization at vnode
  - Better locality for AAE and MDC mechanisms
- Hash each object to it's own location on the ring
  - Improved data distribution
  - Improved parallelization

Retrieval of composed objects

- Strict quorum
  - Reduced availability from the embedded solution
- Sloppy quorums
  - Dangling references
  - Absent references
- Partial writes problematic with either solution

So, where are we?

- Prototype implementation which allows for composition by reference
- Partial failures observed differently:
  - Both susceptible to false-negatives
  - Embedded map converges correctly
  - Reference map orphans objects or applies updates
- How do we handle deferred updates in the map atomically?
- Do we need multi-key atomic transactions?
- Do we need something like RAMP? [1]

Current and Future Work I

- Modify core replication mechanism to ship operations (delta-CRDT)
- Parallel retrieval of referenced objects
- Largely focused on maintaining the map integrity without garbage collection

Current and Future Work II

- Garbage collection
  - Recursive removal of referenced objects
  - Partial write failures; each dependent write could trigger its own series of partial write failures
  - Concurrent removals and additions; how do we know when to clean up all referenced objects when dealing with objects composed with composed objects

References I

P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica.

Scalable atomic visibility with RAMP transactions.

In *ACM SIGMOD Conference*, 2014.

Basho Technologies, Inc.

Riak DT source code repository.

https://github.com/basho/riak_dt.

Boundary.

Incuriosity Killed the Infrastructure: Getting Ahead of Riak Performance and Operations.

http://boundary.com/blog/2012/09/26/incuriosity-killed-the-infrastructur/.

References II

Y. Chen, R. Griffit, D. Zats, and R. H. Katz.

Understanding tcp incast and its implications for big data workloads.

Technical Report UCB/EECS-2012-40, EECS Department, University of California, Berkeley, Apr 2012.

C. Hale and R. Kennedy.

Riak and Scala at Yammer.

http://vimeo.com/21598799.

W. Moss and T. Douglas.

Building A Transaction Logs-based Protocol On Riak.

http://vimeo.com/53550624.