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

(Work in progress report)

Christopher Meiklejohn

Basho Technologies, Inc.

cmeiklejohn@basho.com

Abstract

The Riak DT library [2] provides a composable, convergent replicated dictionary called the Riak DT map, designed for use in the Riak [1] replicated data store. This data type provides the ability for the composition of conflict-free replicated data types (CRDT) [7] through embedding.

Composition by embedding works well when the total object size of the composed CRDTs is small, however suffers a performance penalty as object size increases. The root of this problem is based in how replication is achieved in the Riak data store using Erlang distribution. [4]

We propose a solution for providing an alternative composition mechanism, composition by reference, which provides support for arbitrarily large objects while ensuring predictable performance and high availability. We explore the use of this new composition mechanism by examining a common use case for the Riak data store.

Categories and Subject Descriptors C.2.4 [*Distributed Systems*]: Distributed databases; D.3.3 [*Programming Techniques*]: Language Constructs and Features - abstract data types, patterns, control structures; E.1 [*Data Structures*]: Distributed data structures; H.2.4 [*Database Management Systems*]: Distributed databases

Keywords Dynamo, Eventual Consistency, Data Replication, Commutative Operations, Riak, Erlang

1. Introduction

The Riak DT library [2] provides a composable, convergent replicated dictionary called the Riak DT map, designed for use in the Riak [1] replicated data store. This data type provides the ability for the composition of conflict-free replicated data types (CRDT) [7] through embedding.

Composition by embedding works well when the total object size of the composed CRDTs is small, however suffers a performance penalty as object size increases. The root of this problem is based in how replication is achieved in the Riak data store using Erlang distribution. [4]

PaPEC '14, April 13-16 2014, Amsterdam, Netherlands.

ACM 978-1-4503-2716-9/14/04. . . $15.00.

http://dx.doi.org/10.1145/2596631.2596635

We propose a solution for providing an alternative composition mechanism, composition by reference, which provides support for arbitrarily large objects while ensuring predictable performance and high availability. We explore the use of this new composition mechanism by examining a common use case for the Riak data store.

2. Motivation

Consider a social network application where each user can create events that are visible to other users via a timeline. One way to implement this, that has been used by existing users of the Riak data store [5] [6], has been to model each user's timeline as lists of references to independent objects in the data store, one for each event with a unique key.

We can model this using the Riak DT map by creating a dictionary of entries in the map from timestamps to embedded maps containing the information for each post. Modeling the timeline objects this way has two problems:

- Once objects grow to be over one megabyte, a noticeable degradation in performance can be observed.
- Given Riak does not guarantee causal consistency, it is possible to observe references to objects that are not available. This is known to be true during failure conditions when primary replicas are not available and both read and write operations are handled with sloppy quorums. [3]

Given these limitations, we need a solution for providing an alternative composition mechanism that does not degrade in performance as size increases, but provides values at read time which observe the lattice properties of state-based CRDTs ensuring conflict-free merges with later state.

3. Solution

We explore a solution to the limitations of composition by embedding by proposing the following changes to the Riak data store:

- We extend the existing API as provided by Riak for interacting with the Riak DT map, to support the specification during a write of whether the object should be composed by embedding or by reference.
- When performing a write operation of an object containing references to other objects, we generate unique reference identifiers for each referenced object. Using these unique identifiers, we write the referencing objects first followed by the referenced objects in a recursive manner.

- When performing a read operation of an object containing references to other objects, we recursively attempt to retrieve the referenced objects from the data store.

The above changes are sufficient for providing causal consistency of objects when both the referencing and referenced objects are located across the same set of replicas, however we can not make that guarantee when attempting to provide equal distribution and high availability of data through consistent hashing and sloppy quorums, which is a core tenet of the Riak data store.

3.1 Sloppy quorums and disjoint replica sets

To support sloppy quorums, and the ability to compose objects by reference that span a disjoint replica set, we also need to provide a solution to handle objects that have been composed by reference when the referenced objects are not available. In the event of a referenced object becoming unavailable during a read operation, we can leverage the type information stored in the Riak DT map about composed objects to return the bottom value for the referenced object's type. This ensures that later read operations, where the previous missing reference is now available, correctly merges with the version where it was not.

4. Future work

In this section, we explore work which we believe will improve the performance and viability of this solution.

4.1 Parallel retrieval

Providing a mechanism for parallel retrieval of referenced objects in the map would help increase performance as the breadth of referenced objects increases, as we could launch jobs across disjoint replica sets which run in parallel. We believe that the Riak Pipe processing pipeline, which is used to support Riak's scatter-gather query mechanism would be appropriate for providing the substrate for this improvement.

4.2 Garbage collection

We are still exploring providing a solution for garbage collection of referenced objects. The major concerns of garbage collection arrive from two cases:

- When deleting objects, we need to ensure a recursive removal of all referenced objects. Given that the unique reference identifiers are known by the referencing objects, scheduling these for removal is not problematic. However, knowing how to properly schedule these removals when referenced objects might incur a concurrent update and removal is still unknown as this operation is not safe until the replicas are merged.
- When dealing with a partial failure, we need to ensure that any objects that have been written before the failure are scheduled for garbage collection. For example, when writing an object with three references, we need to make sure that we schedule both the referenced objects for garbage collection as well as the references.

5. Conclusion

In this work, we discuss the challenges involved in implementing this approach and the possible solutions. We explore the drawbacks of composing these values by reference and the problems of garbage collection when dealing with concurrent operations to composed objects.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n^{o} 609551.

References

[1] Basho Technologies, Inc. Riak source code repository. `https://github.com/basho/riak`, .

[2] Basho Technologies, Inc. Riak DT source code repository. `https://github.com/basho/riak_dt`, .

[3] J. Blomstedt. Absolute consistency. `http://lists.basho.com/pipermail/riak-users_lists.basho.com/2012-January/007157.html`.

[4] Boundary. Incuriosity Killed the Infrastructure: Getting Ahead of Riak Performance and Operations. `http://boundary.com/blog/2012/09/26/incuriosity-killed-the-infrastructur/`.

[5] C. Hale and R. Kennedy. Riak and Scala at Yammer. `http://vimeo.com/21598799`.

[6] W. Moss and T. Douglas. Building A Transaction Logs-based Protocol On Riak. `http://vimeo.com/53550624`.

[7] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Rapport de recherche RR-7506, INRIA, Jan. 2011. URL `http://hal.inria.fr/inria-00555588`.