SyncFree Technology White Paper

BigSets: Scaling CRDTs to large sizes in Riak

Russell Brown, Torben Hoffmann
Basho, Inc.

1 September 2016

Sets Scaled to Bigsets

Bigsets [6] make CRDT [10] Sets usable at a scale. The initial implementation of CRDTs Sets in Riak capped the size of a set to the size of an object in Riak. 1Mb sounds like a lot, but for some use cases it just is not enough. Customers have had to resort to partitioning their sets into smaller subsets to overcome this issue. Add to that a drop in write throughput to an unacceptable level well before the 1Mb limit and you have a real problem.

Bigset aims to make Sets scale in both size and write throughput.

The initial transposition of the CRDT papers into Riak was as follows:

1. Create a library of CRDTs (riak_dt [5])
2. Drop the library into Riak via a few hooks
3. Add an API to Riak for acting on CRDTs

As seen in Figure **??** this leads to having the entire CRDT Set inside the Riak object.

This worked fine, in fact bet365 said:

> ...after some analysis we found that much of our data could be modelled within Sets so by leveraging CRDT's our developers don't have to worry about writing bespoke merge functions for 95% of carefully selected use cases...

However, as customers started to use Sets more they found that performance degraded as cardinality grew [8].

This is partialy due to the inherent size limit, but also due to the fact that when we map one key to one Set when a Set is to be mutated it must be read, updated, and written. Read-modify-write of the whole Sets leads to a problem of quadratic bytes-in-bytes out. Even with delta-replication, the whole set must be read-modified-written for each operation. For a database, disk I/O is the most contended resource, and this quadratic bytes-in-out behaviour is unacceptable.

Where to use Bigsets?

Basically all places where the exisiting Sets would be used. Bigsets provide the same functionality as Sets, only better.

bet365 uses sets to keep track of all open bets for a customer [7]. NHS models mailboxes with sets. Another customer is using sets for having a pool of cryptographic keys as well as inverted indexes.

The Tricks of Bigsets

Bigsets is a technique that has been applied to the delta-Opitmised-Add-Wins Set which can also be implemented to other CRDTs. The technique boils down to the de-composition, or splitting up, of the constituent parts of a CRDT and storing them in an ordered way on disk. This means that only the minimum number of bytes need to be read, written, and transmitted over the network. While that technique itself seems fairly trivial, it engenders certain further complexities in implementation: we were dealing with a single key, now we deal with many that are related.

Bigset is an addition to the Riak Data Types [5] feature that uses the de-composed CRDT technique to provide Sets that can be many times larger (millions of elements) with orders of magnitude faster write speed, and a small penalty in read performance for small Sets. However, this cost is vastly offset by Bigset's ability to support a queries without reading and transmitting the fullset. Queries include: subset, range and matches. Bigset also supports stream results, with pagination.

Using Bigsets

Bigsets have the same user experience as the current Riak Data Type Sets. The client API is operations based. The client sends "add" and "remove" operations to update the set. And queries the set for subset, membership, ranges. Results can be streamed and paginated. In summary, as long as the subsets being merged are an equal subset of the whole set, they can be merged correctly since the set of events covered by the version vector is extrinisc to the events for the subset. This means that *any* equal subsets from different replicas can be merged. Opening the way for streaming reads and other queries.

Bigsets in relation to the other SyncFree producs

Bigsets is different to the other SyncFree projects in that it has truly focussed only on the productisation of CRDTs. There are no new guarantees, no new data types. We've taken the technology and made it scale and perform well enough for use in a commercial setting. And we've taken advantage of delta-replication, plus we've invented some novel things:

- one sided full state merge based handoff which allows us to merge one replica with another, whilst only reading the data from one side. A huge IO saving
- incremental merge, which enables streaming, queries, and pagination

Both, we hope, will feed into the other SyncFree projects.

As far as semantics go this is the same delta-replicated optimised Add Wins Set from https://arxiv.org/abs/1210.3368 in 2012, with added deltas from https://arxiv.org/abs/1603.01529. The innovations are engineering "tricks" that make the set scale.

Bigsets in relation to other products

The closest by appearance to bisgets is maybe REDIS [1]? When we shipped data types in riak 2.0 they were immediately compared to REDIS, and there has been talk of sticking a REDIS API on Riak Data Types. In terms of performance, Cassandra might be seen as a similar well-known product, except it uses the lossy Last Write Wins reconciliation, where an arbitary write is chosen as the "correct" or winning value based on timestamp, and the rest lost.

There is no Convergent Replicated Data Types backed datastore on the market, except Riak.

Competitors to Bigsets

There still is *NO* commercial product backed by CRDTs except Basho's Riak, and Bigset is another step forward for Riak. With this differentiator, there is no NoSQL database that can directly compete with Riak. When comparing another NoSQL database to Riak they instead fall into one of two categories:

1. The new SQL type which trade off availability for consistency, like LMDB.
2. The plain Key->Value data base, like Cassandra.

Riak doesn't directly compete with the former (though bigset maybe an enabling technology for eventually consistent SQL tables), and differs from the latter in that it does not lose concurrent writes with similar timestamps.

How does Bigsets do its magic

Previous releases of Sets in Riak Data Types mapped a single key to a single set in a riak_object on disk.

As mentioned above the primary technique is to break the CRDT Set into its constituent parts and store them independantly, but co-located, on disk.

Bigsets engineers the CRDT into a database. Riak Data Types takes a library of state-based CRDTs and stores instances in a database.

Bigsets is backed by leveldb [2]. Leveldb [3] is a key-value database that stores keys in sorted order. We hash all Set elements by Set name and store them together, in order, on disk.

Whereas before, in order to add or remove an element from the set, the whole set had to be read from disk, with this scheme only the version-vector [9] for the set needs to be read at update time. Adding a new element only requires updating the version vector to get a causal tag [4]. The new element and updated version vector are written. Only the new element and its causal tag are replicated. For removes, the client sends the element to be removed and its causal tags, these are added to the clock (if unseen) and deletes generated. Again, no need to read the whole set as the elements are independant of one another.

This leads to insert times that depend on the size of the Version Vector only, not the size of the whole set.

This multi-key approach also enables querying. Whereas before to answer questions about a set (is X a member, is [X,Y,Z] a subset etc) the whole set must be read, with bigset, we can read the range of keys needed to answer the query. For membership checks this is very efficient.

As the keys for an element are stored in order in leveldb for a subset query (of which a membership query is the smallest, a subset of one) we read the clock, we seek then to the first key in the query subset and read it, and we iterate, seek, iterate, seek until the last key in the range. This is significantly faster than reading the entire set and iterating it in memory to find the set intersection. Consider asking if [X] is a subset of a 10 million element set, for example.

To read subsets of a set across a quorum of nodes we need to be able to merge divergent subsets. Bigsets can do this because it operates on matching ranges of the keys set from different replicas. For example, for membership query, each replica reads the keys for the member and sends them to a query co-ordinator,

The query co-ordinator performs a normal CRDT Set merge on that subset. Since the subsets are of an identical range the merge works even though the clocks contain information for a much larger set, due to this extra information being extrinsic to the subset elements.

In summary, although the scheme of playing to the strength of the storage layer, and breaking the Set into small consituent parts seems trivial, it is a step away from the library apporach that has pervaded all CRDT systems research. This step has yielded benefits of both scale and performance,

Conclusion

Sets, not only the CRDT Sets, have always been a nice abstraction to use in programs, but using them often requires trade-off due to the nature of the underlying data structures.

Bigsets expands the range of applications where Sets can be used as it provides the ability for straightforward and effective scaling. While there is a slight penalty in read performance, the scalability and performance gains for key Set operations such as subset, range and matches significantly outweighs any downside making this a real positive development for anyone dealing with these issues.

References

[1] https://redis.io/.

[2] https://leveldb.org.

[3] https://github.com/basho/leveldb/.

[4] Paulo Sérgio Almeida, Carlos Baquero, Ricardo Gonçalves, Nuno Preguiça, and Victor Fonte. *Scalable and Accurate Causality Tracking for Eventually Consistent Stores*, pages 67–81. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[5] Basho Technologies. Riak data type library, 2011–2016. https://github.com/basho/riak_dt).

[6] Russell Brown, Zeeshan Lakhani, and Paul Place. Big(ger) sets: decomposed delta CRDT sets in riak. In *PaPoC'16: Proceedings of the 2nd Workshop on the Principles and Practice of Consistency for Distributed Data*. ACM, 2016. https://dl.acm.org/citation.cfm?doid=2911151.2911156.

[7] Dan Macklin. Key lessons learned from transition to nosql at an online gambling website, 2015. https://www.infoq.com/articles/key-lessons-learned-from-transition-to-nosql.

[8] Kyle Marek-Spartz. Benchmarking large riak data types: A potential fix, 2014. http://kyle.marek-spartz.org/posts/2014-12-03-benchmarking-large-riak-data-types-a-potential-fix.html.

[9] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Trans. Softw. Eng.*, 9(3):240–247, May 1983.

[10] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Research Report RR-7506, Inria – Centre Paris-Rocquencourt ; INRIA, January 2011.