Eventual Consistency and Deterministic Dataflow Programming

Christopher Meiklejohn

Basho Technologies, Inc. / SyncFree Consortium

LADIS Workshop ’14
Cambridge, United Kingdom
October 23, 2014
Overview

1. Motivation
2. Our Approach
3. Conclusions and Future Work
4. References
Motivation

- Riak [7] is a Dynamo-inspired [4] key-value store
- Querying by key main mechanism for data storage and retrieval
- Three mechanisms presently for more expressive data access:
  - MapReduce-like system [3, 6], secondary indexing, integration with Apache Solr
- Each additional mechanism contains drawbacks
  - Mechanisms are not fault-tolerant
  - Structure is rigid: need to know schema when storing data
  - Not possible to perform composition of queries
Building large scale distributed applications with strong properties

Users want to be able to compute with their data in an efficient and composable manner while guaranteeing strong properties

Provide a framework for building eventually consistent materialized views that have strong convergence properties

- Data types which have strong convergence properties
- Deterministic language to compose data types and preserve these strong convergence properties
Our Approach

- Leverage conflict-free replicated data types (CRDTs)
- Builds on top of Derflow [1], a deterministic dataflow language
- Extends Derflow with CRDTs, similar to LVars [8] and Bloom$_L$ [2]
- Applies techniques from C-CRDTs [10] for incremental computation
- Uses version vectors with exceptions for tracking program state [9]
- Uses Dynamo-style quorums for aggregating computation results
CRDTs

- Conflict-free Replicated Data Types [11]
- Strong Eventual Consistency (SEC)
- Data structures which contain resolution logic
- Based on mathematical properties

\begin{figure}
\centering
\begin{tikzcd}
\emptyset & \text{add}(e) & \{e\} & \text{rmv}(e) & \{e, \varnothing\} & D & \{e, \varnothing\} \\
\emptyset & \text{add}(e) & \{e\} & D & \{e\} & D & \{e, \varnothing\}
\end{tikzcd}
\caption{Set where element can be added and removed once. \textsuperscript{1}}
\end{figure}

\textsuperscript{1}Can be generalized to arbitrary adds and removes.

Distributed deterministic programming language
Similar to Mozart/Oz, Ozma [5]
Relies on a distributed variable store
Extend the model to state-based CRDTs and C-CRDTs
Inputs and outputs of programs are CRDTs
Computational CRDTs

- Defines CRDTs which observe partial state
- Allows for chaining, but requires explicit mapping function
- Defines a merge function to combine partial results
- Maps well to Dynamo data storage model
  - Replicas within a replica set should have equal state,
  - Disjoint between replica sets
Version Vectors with Exceptions

- Each program tracks the versions of inputs used in computations
- Data is tracked in a version vector with exceptions
- Treat computations as values in Dynamo-style system
- Can be used for anti-entropy, read repair, etc.
Dynamo Quorums

- Identify covering set of nodes for key space
- Can include multiple copies from the same replica set
- Merge (LUB) across replica sets
- Merge (Delta) across copies of the same replica

Figure: Ring with 32 partitions and 3 nodes
Dynamo Quorums

- Identify covering set of nodes for key space
- Can include multiple copies from the same replica set
- Merge (LUB) across replica sets
- Merge (Delta) across copies of the same replica

Figure: Ring with 32 partitions and 3 nodes with covering set identified in red

Meiklejohn (Basho)
Derflow
LADIS ’14
Dynamo Quorums

- Identify covering set of nodes for key space
- Can include multiple copies from the same replica set
- Merge (LUB) across replica sets
- Merge (Delta) across copies of the same replica

Figure: Ring with 32 partitions and 3 nodes with fault-tolerant covering set identified in red and blue
Conclusions and Future Work

- **Current status:**
  - Derflow\textsubscript{L} implemented on Riak Core, extended with CRDTs
  - Implemented a test harness for Derflow\textsubscript{L} using open source work from Basho to test distribution and operation in partitions

- **Future work:**
  - Composing CRDTs currently is very explicit and cumbersome
  - Will be solved as part of the SyncFree research project
  - Program analysis for detection of incorrect programs
M. Bravo, Z. Li, P. Van Roy, and C. Meiklejohn.
Derflow: Distributed deterministic dataflow programming for erlang.

Logic and lattices for distributed programming.
Technical Report UCB/EECS-2012-167, EECS Department, University of California, Berkeley, Jun 2012.

J. Dean and S. Ghemawat.
Mapreduce: Simplified data processing on large clusters.

Dynamo: Amazon’s highly available key-value store.


S. Doeraene and P. Van Roy.

A new concurrency model for scala based on a declarative dataflow core.

In *Proceedings of the 4th Workshop on Scala, SCALA ’13*, pages 4:1–4:10, New York, NY, USA, 2013. ACM.

B. Fink.

Distributed computation on dynamo-style distributed storage: Riak pipe.

R. Klophaus.
Riak core: Building distributed applications without shared state.

L. Kuper and R. R. Newton.
Lvars: Lattice-based data structures for deterministic parallelism.

D. Malkhi and D. Terry.
Concise version vectors in winfs.
D. Navalho, S. Duarte, N. Preguiça, and M. Shapiro.
Incremental stream processing using computational conflict-free replicated data types.

M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski.
A comprehensive study of Convergent and Commutative Replicated Data Types.