Entwicklung eines Datenbanksynchronisations-Mikroframeworks auf Basis konvergenter, replizierter Datentypen

Sebastian Götte und Matti Möll

27.10.2014

1 History of CDRTs and the current state of research

Data synchronisation has been an important area of research for several decades. Recently, discourse has been heightened due to increasing parallelism of contemporary system architectures. Traditionally, such systems have either relied on a central authority[1] or been largely limited to read-only affairs[2].

Well-defined automatic merge strategies to reconcile diverging realities in different network locations have been in devlopment for some time now, especially under the aegis of distributed version control systems (DVCSs) such as git, Mercurial and Darcs but have not yet seen widespread deployment.

One possible solution to this problem is what we call Convergent Replicated Data Types (CRDTs). CRDTs have been used by computer scientists for at least 40 years. The concept was only formalized and its name coined five years ago in [4]. Since then, there has been some further research into the formal properties of CRDTs[3] and first real-world implementations of the concept using the term *CRDT* have appeared[4].

In 2007, Amazon.com published a paper detailing the architecture of their *dynamo* key-value store. From a technical standpoint, Dynamo's replication system is behaving like a CRDT set implementation.

We want to give an overview of the history and existing implementations of CRDTs followed by an overview of some concepts that may be used in conjunction with CRDTs to provide useful higher-level behavior and end with some exemplary possible real-world use cases.

2 Convergent replication

CRDTs have first been defined in [4]. [4] are using the term *CRDT* for *Collision-free Replicated Data Type* and are distinguishing between *CvRDTs* (*Convergent Replicated Data Types*) which they are also calling *State-based CRDTs*, and *CmRDTs* (*Commutative Replicated Data Types*) which they are also calling *Operation-based* or *Op-based CRDTs*. Since both concepts are shown

[1] See every RDBMS in existence, OpenLDAP and ActiveDirectory, RSS and apt among others
[2] See e.g. BitTorrent and other classical file sharing systems
[3] See [5]
[4] See e.g. [6] resp. [8] and [7] and [2]

to be exactly equivalent, we are using the acronym *CRDT* for either approach and settled on the long form *Convergent Replicated Data Type* since it is describing both very well and is sufficiently handy.

2.1 Operation-based CRDTs

An op-based CRDT is a data type whose value is defined solely by a set of commutative operations applied to a common base state. For illustration, consider an up/down counter as a simple example. The initial state would be 0 and the operations would be to add or subtract a number. The current counter value is defined as the sum of all add/subtract operations performed on this instance so far. Since addition is commutative, the order of these operations does not matter. In a distributed system, additions and subtractions on the same counter can be performed simultaneously on multiple nodes, and the resulting conflict can be resolved by each node telling each other node all operations that have been performed on its instance.

2.2 State-based CRDTs

A state-based CRDT is a data type which is associated with a partial ordering on its value space. A simple example for a state-based CRDT is an add-only set. The associated partial ordering is the subset relation. The merge operation is the set union.

State- and Operation-based CRDTs are equivalent in that each can be used to emulate the other. Most existing implementations use a state-based storage, where some do incorporate some op-based-like behavior for more efficient sync[5].

3 Non-exclusive roadmap for this project

3.1 Real-world adaption

We assume the adaption issue is most likely to be solved by a *useful* API allowing it to possibly be integrated into existing systems, on top of an existing database. The API issue is both one of the engineering side and of the user interface side, which makes it just a little more complicated since one does not often see a solution satisfying both concerns in real projects.

We decided on tackling this issue first as a solution to it serves as a foundation for any further work.

In our opinion, a properly implemented solution would be kept as simple as possible, being but a maximally flexible, lightweight microframework, serving as a shim on top of an existing database layer. This would system would only provide the elementary protocol logic and synchronization "magic" in an attempt to not encumber the user with a variety of features better implementations of which can be found in any number of other, *proper* frameworks.

3.2 Application in untrusted networks

CRDTs are a useful building block for a distributed system, however they pose one major problem: By nature, they tend to require monotonically increasing amounts of memory. This can be alleviated, but not completely solved by means of garbage collection. In the context

[5] e.g. Amazon Dynamo is computing a delta using a Merkle tree and then only transmitting the subset of changed entities, which is equivalent to the set of operations since last launch.

of a distributed system this means that any untrusted, large-scale architecture will be under a serious risk of resource exhaustion attacks. The integration of trust and federation in an untrusted large-scale network is an important topic for future research.

One might for example conceive a Kad-like DHT being used for node discovery with a web-of-trust based verification process layered on top to prevent certain types of attacks relying on fake network identities.

If there is any time left after finishing the aforementioned microframework to a satisfactory degree, we want to explore this angle as much as possible for *this* is where all the *interesting* problems live.

4 Possible use cases

...accompanied by some subjective classification for interestingness

Collaborative document editing: Though this one seems to have attacted quite some attention in the past, we consider it sub-par revolutionary since it seems collarborative editing schemes always have a certain smell to their code bases.

Distributed microblogging, or, depending on the choice of parameters and resulting balance between system *capacity* and *latency*, macroblogging: We think this might have some merit, since for one people seem to actually be using systems of this kind, and it seems kind of simple on the surface.

Distributed public discussion, or, *derrit* as we want to call this, meaning a distributed platform that can be used to exchange public comments in discussions, sorted by theme and the comments' semantic relationship. We would ultimately like to have this system, but as of now we consider it too complex to be implemented on a whim.

Distributed just-another-android-apps: About 50% of our group were found to be interested in apps doing ordinary things like asking the user questions from virtual flash cards being able to synchronize between devices without regular crashes and data loss. We think this option might be worthwhile to follow up on in the future, but for now we will just try to convince our friends and distant relatives to work on these due to a lack of lifespan and complex eclipse installation procedures.

Literatur

[1] Scott Aaronson. Eigenmorality. `http://www.scottaaronson.com/blog/?p=1820&re=1`, 2014. [Online; accessed Oct 27 2014].

[2] Peter Bourgon, Tomás Senart, Björn Rabenstein, and Johan Uhle. Roshi: a crdt system for timestamped events. `https://developers.soundcloud.com/blog/roshi-a-crdt-system-for-timestamped-events`, 2014. [Online; accessed Oct 27 2014].

[3] DeCandia, Hastorun, Jampani, Kakulapati, Lakshman, Pilchin, Sivasubramanian, Vosshall, and Vogels. Dynamo: Amazon's highly available key-value store. 2007.

[4] Mihai Letia, Nuno Preguiça, and Marc Shapiro. Crdts: Consistency without concurrency control. 2009.

[5] Marc Shapiro, Nuno Preguica, Carlos Baquero, and Marek Zawirski. A comprehensive study of convergent and commutative replicated data types. 2011.

[6] Basho Technologies. Riak. `http://basho.com/riak/`, 2009. [Online; accessed Oct 27 2014].

[7] Basho Technologies. Distributed data types in riak 2.0. `http://basho.com/distributed-data-types-riak-2-0/`, 2014. [Online; accessed Oct 27 2014].

[8] Basho Technologies. Riak 2.0 announcement. `http://basho.com/riak-2-0-new-capabilities-new-use-cases-available-for-download/`, 2014. [Online; accessed Oct 27 2014].