Something about CRDT application

E. M. Bongers

Something about CRDT application

Master's Thesis in Computer Science

Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

E. M. Bongers

25th October 2020

Author
E. M. Bongers

Title
Something about CRDT applications

MSc presentation
25th October 2020

Graduation Committee
ir. M. de Vos (supervisor) Delft University of Technology

Abstract

TODO ABSTRACT

Preface

TODO MOTIVATION FOR RESEARCH TOPIC

TODO ACKNOWLEDGEMENTS Thanks to M. de Vos for getting stuck with me.

I will do everyone a favour and promise to not pursue further academic advancement after this.

E. M. Bongers

Delft, The Netherlands
25th October 2020

Contents

Chapter 1

Introduction

TODO INTRODUCTION

TODO ORGANISATIONAL DESCRIPTION OF THESIS

Chapter 2

Background & State-of-the-Art

TODO: Intro TODO: Organization

2.1 Distributed Systems and Central Components

Distributed systems refer to a group of computers where each computer does a part of the work (computing/storage/communication/etc.) in service to the collective in order to achieve some end goal. Many such systems exist in the world with http (the world wide web) being a prime example and the domain name system (DNS) another. However these distributed systems rely on centralized components (such as central servers) to perform some critical function, and such centralized components are a weak point for a distributed system.

Centralized components have three aspects that make them undesirable in any distributed system. First of al central components are a single point of failure, so part of the service cannot be provided if they are unreachable or unresponsive. Secondly, they are inevitably controlled by a single entity, and any entity is influencable by private actors and governments. The worst case being manipulation of the service provided by the central component. This is generally impossible to detect and can severely affect the functioning of a distributed system. Thirdly, central components also have a monetary cost associated with them and the distributed system user community has to provide for this in some way.

Often the use of central components is a design compromise. It is often easy to have a (part of a) process execute in a controlled and trusted environment. This simplifies designs greatly. A good example of this are MMO games, a distributed system where the actual game simulation happenes in a controlled environment on a central server cluster. However distributed gaming without a central component has yet to be perfected, thus for some designs it might be the only practical choice.

While central components can be used to great effect, they are to be eschewed in distributed systems that wish to operate reliably in the face of adverse conditions. The resulting distributed systems are called peer-to-peer systems, each participant

is an equal peer[1].

2.2 Synchronization

Explain about the CAP theorem [1].

As CRDT paper points out, Partition is always going to happen at internet scale, someone will have a bad link somewhere. So any system will need to account for that. Availability is paramount because of user expectation. So they throw Consistency under the bus and aim for Strong Eventual Consistency. (TODO: clean this)

2.3 Operational Transformation

Explain about OT as proposed by [2]

Explain what it's limitations are. Why do we want CRDTs? Are the better?

2.4 Conflict-free Replicated Data Type

To solve the synchronization issue several solutions have been proposed, a recent paradigm is that of Conflict-free Replicated Data Type (CRDT) as described in [3]. The idea of a CRDT is to achieve strong eventual consistency of a replicated data-structure by structuring data and updates in such a way that no conflicts can arrise. This then allows trivial automatic merging of different versions or updates of the data structure. In terms of the CAP theorem this provides Availability and Partition tolerance, but with a strong formal guarante that Consistency will be reached eventually.

The description of CRDT in [3] provides two formal models for reasoning about CRDTs, the state-based Convergent Replicated Data Type (CvRDT) and Op-based Commutative Replicated Data Type (CmRDT). The two models are equivalent but (apparently) CvRDTs are convenient for formal reasoning and CmRDTs are more convenient in implementations.

The CvRDT model is based on a join semilattice, in this case it means a partial ordering on the set of all states held by all replicas, and a function (known as 'join' or 'least upper bound') for pairs in the set. The join function produces a state that orders strictly greater than its two inputs and is commutative, idempotent and associative. A further requirement is that if a pair of elements from the set order equally ($x \leq y$ and $y \leq x$) then $x \equiv y$. Together these conditions imply that states can always be joined, and the result monotonically proceeds up the partial ordering chain. It must therefore converge to a maximal element in the partial ordering, one where all initial states have been joined, and because of the equivalence relation any maximal element will do. Although in practice the maximal element could well be

[1] and no peer is more equal than others

singular, the greatest element of the partial ordering. The CvRDT definition of a CRDT permits testing of a data type to determine if it is a CVDT, but it leaves a lot (everything) to the imagination when it comes to designing one. Lets see if the CmRDT definition is more practically usefull.

The CmRDT model assumes a reliable causally ordered broadcast communication protocol and uses that to deliver state update operations to all replicas. To each state update operation is bound a side-effect free precondition test that determines if an update operation may be applied. If at a replica two updates are pending, i.e. their preconditions satisfied, applying either update may not invalidate the preconditions of the other. This allows updates to be executed in any order once their preconditions are met, or put another way, all concurrent operations must be commutative. This ensures that updates can always be applied eventually, and thus lead to an equivalent state at each replica.

Within the study of CRDTs there are several basic data structures that are known, such as a vector clock, monotonic counters and add only sets. (TODO: find all the others) Compsitions of these basic datastructures develops more advanced data structures. For example using two monotonic counters I and D its possible to create a non-monotonic counter by computing I - D. But more complex compositions allow for sets, dictionaries and directed graphs.

Thoughts about crdt:

- They lead to full replication of an objects state. All the nodes are eventually aware (and incorporate) all the updates from the other replicas. This is somewhat at odds with (infinite) scalability, growing groups attempting to synchronize a single data structure will falter at some point. One idea to push this back somewhat is to deliberately partition the network and only have the partitions merge every so often. But this is still a lot of state to keep around. Especially things like a vector clock don't scale at all well.

- There is a "test" for what constitutes a crdt, that means that there are multiple basic crdt structures, that can be composed in new ways. I think many will have sought for them and described them in literature. There could be real gems in there.

- CRDTs have applications outside of peer-to-peer systems, they might be usefull in parallel computing. They are also usefull for databases since they explicitly aim for eventual consistent state.

- TrustChain is in some way a CRDT, the blocks form a partially ordered set (full block (A, B) as a subset of all blocks, and is dominated by any fullblock containing A+1 or B+1. Question to awnser, is there a join, a least upper bound to full block (A, B)? I think there might be multiple. But also TrustChain doesn't have a true update function either. Its an add only set.

2.5 TODO Priority queue

Kleppmann's Local First "manifesto" for crdts looks like the sort of angle I could aim for, big tech sucks, crdts to the rescue!

Merkle-CRDTs paper is interesting, since it promises to do something about the clock that is needed in some crdts. It also scales to "upto thousands" of clients. Is that good enough for practical situations or should the aim be milions?

Vagvisir and FabricCRDT look slightly less interresting since TrustChain is already pretty close (if not equivalent) to a CRDT. Will check if they add anything, but I doubt it.

There are a lot of projects that combine crdts with ipfs, i'm not sure of our lab's feelings on ipfs, and also what the reason is that this combination is attractive.

Bibliography

[1] A. Fox and E. A. Brewer, “Harvest, yield, and scalable tolerant systems,” in *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, pp. 174–178, 1999.

[2] C. A. Ellis and S. J. Gibbs, “Concurrency control in groupware systems,” in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’89, (New York, NY, USA), p. 399–407, Association for Computing Machinery, 1989.

[3] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, “Conflict-free replicated data types,” in *Symposium on Self-Stabilizing Systems*, pp. 386–400, Springer, 2011.