Conflict free p2p replicated datatypes

Kim Hammar
kimham@kth.se

Maxime Dufour
maximed@kth.se

May 22, 2017

1 Introduction

Conflict Free or Commutative Replicated Data Types (CRDTs) are special data types designed to support concurrent operations on replicated data without risking conflicts or requiring explicit upfront synchronization. A casual description of CRDT is that they enable consensus-free replication. More formally, a CRDT is a data type where all concurrent operations commute with one another [2].

This report presents an implementation of a P2P system for collaborative editing as part of a course on Distributed Computing and P2P systems. The system is based upon an implementation of the Logoot-Undo CRDT[3] and a causal broadcast implementation. Furthermore, the implementation uses a specific peer-sampling service called Croupier[1]. Croupier is used to implement a Gossiping Best Effort Broadcast, which the causal broadcast depends on. A significant part of the implementation consists of simulation scenarios to demonstrate the system and obtain confidence in the correctness of the implementation.

The system is implemented in Java using Kompics framework[1]. The source code for the implementation can be found on GitHub[2]

2 System Overview

2.1 Assumptions

We assume the lower level networking protocol TCP in combination with IP-routers in the network to provide a perfect-link abstraction. All our network-based algorithms assume access to the perfect-link abstraction.

2.2 Network Topology

The system follows a typical peer-to-peer network topology where each node has a partial view of the network with a set of neighbors c, where $1 \leq c \leq N$ and N being the size of the network. In the figure below an instance of the system with $N = 8$ and $c = 2$ is depicted. As mentioned, each peer uses a peer-sampling service, Croupier, which will provide each peer with a new set of neighbors periodically.

Each peer follows a very simple behavior. After performing an edit to the document, the peer will broadcast the edit-operation to all of its current neighbors. When receiving an edit from someone else, a peer will apply it to its local replica of the document. The guarantees of the Logoot-Undo CRDT in combination with the causal-delivery guarantee of the broadcast abstraction, ensures that all editions will commute when applying it to the local replicas. In figure 1, one of the nodes is broadcasting an insertion with the line "content" into the document to its neighbors. In addition to updating its local replica when receiving an edit from another peer, peers will also make sure that all of their neighbors have received the update by relaying it to them, this functionality is built-in to the broadcast abstraction.

[1] http://kompics.sics.se/current/index.html
[2] https://github.com/62maxime/ID2210_Peers-To-Peers

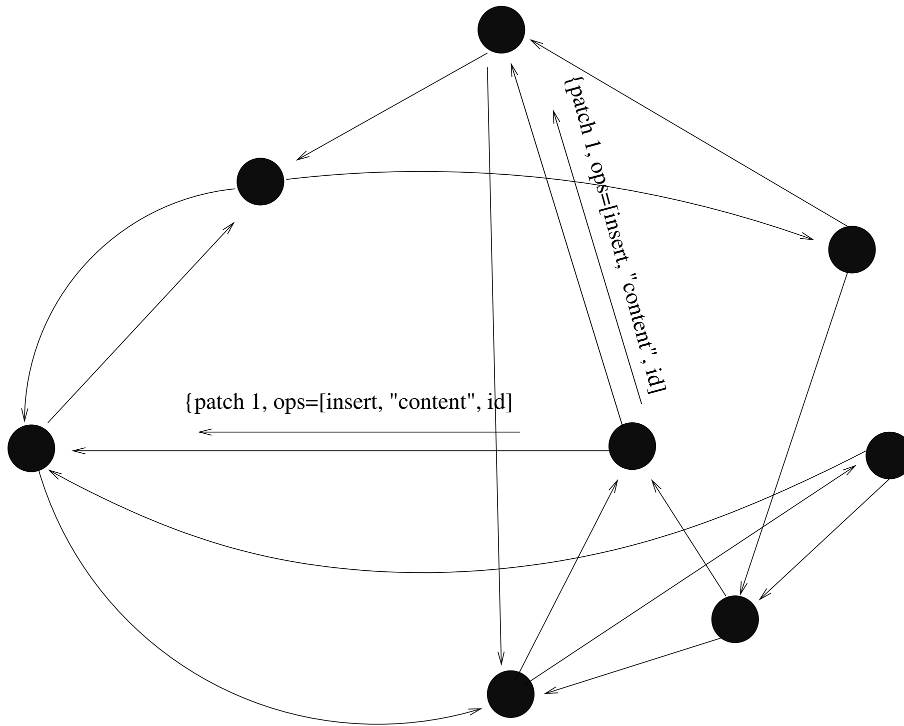

Figure 1: Peer to Peer architecture

3 Implementation

3.1 Component Hierarchy

Figure 2 depicts the hierarchy of Kompics components at each peer. Croupier port and Network ports are provided by the skeleton code. AppComp uses its CausalOrderReliableBroadcast port to broadcast updates of the Logoot-Undo document to other peers. Croupier provides the GBEB component with periodic uniform samples of other peers in the system, which enables broadcast of updates through the network without requiring each peer to have a complete view of all peers in the network.

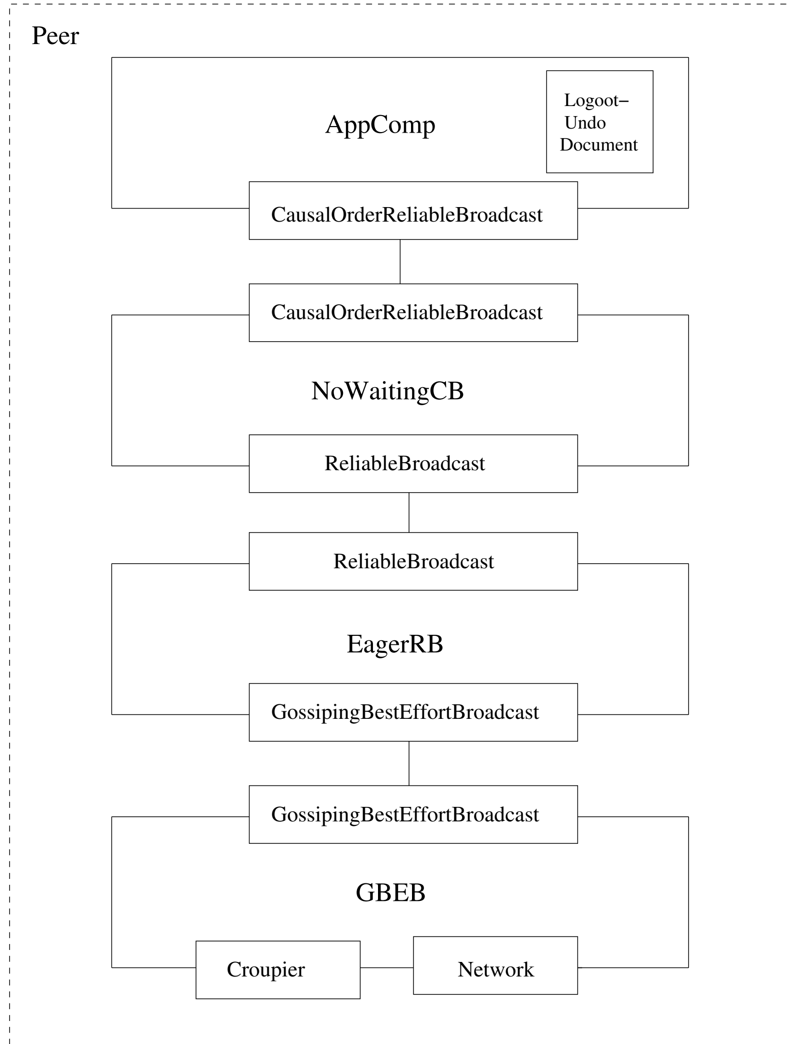

Figure 2: Kompics component hierarchy of a peer

3.2 Logoot-Undo

In `src/main/java/se/kth/app/logoot` the classes and algorithms implementing the Logoot-Undo solution explained in the paper[3] is located. The Kompics component editing a logoot document (`AppComp.java` in our case) will handle delivery of *Patch*, *Undo* and *Redo* events as well as store the Logoot-Undo-Document itself.

Document is the main class of our Logoot-Undo implementation, it keeps all information required for the Logoot-Undo algorithm:

- *site* which is an Identifier that uniquely identifies a site
- *documentLines* which stores the real document in an ArrayList<String>
- *idTable* which stores the corresponding identifier of a document line
- *hb* which stores all patches delivered in a HashMap<UUID,Patch>
- *cemetery* which stores all lines that should not be visible in a HashMap<LineId,Integer>

4 Paper Review. *Logoot-Undo: Distributed Collaborative Editing System on P2P networks[3]*

4.1 Motivation

Concurrent editions in traditional collaborative editing systems might result in conflicts that are not easily merged. CRDT (Commutative Replicated Data Type) is a category of algorithms that proposes a different solution to this problem where it is required that all operations commute. As all operations commute, no conflicts due to concurrent operations can happen and complex merge functions are not needed.

Being able to undo operations in a collaborative editing system is evidently very important and used feature in today's editing systems. Previous class of CRDT algorithms for collaborative editing have focused on supporting insert and delete operations but have lacked the undo feature. The authors motivate their work on the Logoot-undo CRDT algorithm supporting "undo anywhere, anytime" feature by recognizing that the undo feature is commonly a user-required feature, a feature useful to recover from unexpected results that can occur during concurrent editing and a way to deal with vandalism acts in open editing systems such as Wikipedia. Further more, previous work on collaborative editing systems that do support the undo feature is based on other ideas than the algorithm proposed in this paper.

4.2 Contributions

The main contribution of this paper is a CRDT algorithm, Logoot-undo, that apart from insert and delete operations also support the undo feature. The proposed algorithm is a CRDT algorithm and integrates the undo feature with the insert and delete operations as a commutative operation that is suited for P2P systems with a high degree of churn. Additionally, the paper contributes with the following:

- Correctness proof of the algorithm
- Evaluation of the algorithm compared to previous work that demonstrates that the proposed algorithm is an improvement to previous solutions and demonstrates good performance of edit operations.
- A new strategy for generating Line-Identifiers based on boundaries to reduce the pace in which line identifiers grows in the document.

4.3 Solution

The authors propose an algorithm, Logoot-Undo, which belongs to a framework of algorithms implementing logical data structures referred to as "CRDTs" by previous related work. This entails that the algorithm is based on operations that are naturally commutative and don't require synchronization and complex merge operations. An essential part of the commutativity of insert and delete operations of the Logoot-Undo algorithm is the line-identifier which by the algorithm is guaranteed to be unique and totally ordered. This property of the line-identifier ensures that concurrent insert-operations commute. The undo feature is based on an idea of "degrees". Firstly, each patch of operations will have a degree indicating if the patch has been part of an undo, a redo, multiple undoes, etc. Secondly, each line in the document will be associated with a visibility-degree determining if the line is visible or not in the document, this visibility-degree is especially important in case of distinct and concurrent undos/redos affecting the same line. The paper proposes a data structure referred to as "Cemetery" for maintaining this degree information.

The strategy for generating line-identifiers is based on a concept of boundaries. The algorithm for generating line-identifiers aspire to keep the growth in size of line-identifiers more manageable than previous approaches which typically used the random-strategy. The boundary approach uses a given boundary to limit the distance between two line identifiers.

Conducted evaluations are based on different types of pages of the Wikipedia editing system and compares the proposed algorithm with previous work in terms of identifier-generation strategy and performance. The evaluation uses an open API of Wikipedia to obtain XML files and then a diff-algorithm to compute modifications between different versions of the document.

4.4 Strong Points

- The solution tries to be as generic as possible in terms of implementation and makes minimal assumptions.
- The complexity analysis of the proposed algorithm and previous work is not comparable since the structures are different and the complexity result depends on different parameters. The authors therefore make sure to provide the reader with the possibility to compare the algorithms by conducting extensive experiments and evaluations with concrete results that are comparable. The experiments give high credibility due to the choice of documents to evaluate on (Wikipedia) and selection of pages with high diversity for the tests.
- The authors prove the correctness of their work.

4.5 Weak Points

- As the results are based on an average of 10 runs, presentation of the distribution of the results or the standard deviation is missing.
- Conclusions that can be drawn regarding the performance of Logoot-Undo compared to other collaborative editing systems that support the undo feature is based on an assumption that the number of concurrent editions is low. The evaluations give good indications of the strong points of the Logoot-Undo algorithm but it is restricted to a single editing system which exhibits limited number of concurrent edits which is of benefit for Logoot-Undo while not as beneficial for the other algorithms that was compared with. This is due to the fact that one of the data-structures used in Logoot-Undo, the Cemetery, is explicitly designed to have low overhead in systems with few concurrent editions. Evaluations on editing systems with different characteristics might give different results.

5 Implementation experience

We implemented the complete Logoot-Undo algorithm with associated causal reliable broadcast in Java and simulated it in a peer-to-peer system.

Overall the Logoot-Undo algorithm was very smooth to implement in Java based on the pseudo-code presented in the paper as well as general implementation suggestions provided. However a few things that we noted during the implementation that might be worth noticing for other people looking at implementing the Logoot-Undo algorithm:

construct(r,p,q,site) function

The `constructId(r,p,q,site)` function uses a for-loop iterating over the digits of `r`, where r is a number in between the digits of p and the digits on q. In the paper there is not mentioned that getting the correct digit for each iteration of the loop require some work, at least in Java. In the paper it is simply written $d := i^{th}$ digit of r. Since r is just a number (an integer in our Java implementation) the first intuition when implementing the pseudo code is to convert the number to a string and extract the ith digit in the string, doing so is not right since digits in this context does not necessarily have to be in the range $0 - 9$. An example:

$p = \langle 2, 4, 7 \rangle \langle 59, 9, 5 \rangle$
$q = \langle 10, 5, 3 \rangle \langle 20, 3, 6 \rangle \langle 3, 3, 9 \rangle$
$r = 102001$

I.e the correct way to do the for-loop is to realize that the digit-size of r is 3 and the digits extracted in the loop is as follows, first digit: 10, second digit: 20, third digit: 1. To do this in our implementation we implemented a helper function `getDigits(r, p ,q)` that based on the digits of p and q as well as the knowledge that r should be between p and q builds a list of the digits of r that then can be accessed in the for-loop.

Gossiping Best-Effort-Broadcast When implementing this component, we have noticed that the pseudo-code given has one bug, peers do not deliver their own messages. Indeed, after receiving the broadcast request by the upper-layer, they put the message in the past set.

However, when they want to deliver messages, they deliver every message minus the ones in the past set. To solve this problem we just trigger the Deliver event before adding the message to the past set.

6 Simulation and Evaluation

Simulations were conducted with Kompics simulation framework. The tests mainly aim to verify the claimed correctness properties for each component/algorithm, e.g. causal delivery, reliable delivery, commuting editions to Logoot-document etc.

All the tests follow the same principle idea. Each test has a designated scenario located in `/src/test/java/se/kth/tests/testname/sim/scenarios/`. The scenario typically will start a bootstrap server, a number of peers and an observer-component. Some scenarios will also explicitly simulate churn by killing and restarting nodes. Since each test simulates different properties and behaves differently, the tests commonly are associated with a few components that are slight derivation of the original components and include for example saving local-state or simulating crashes, these components can be found in `/src/test/java/se/kth/tests/testname/sim/components/`. During the test the peers will typically perform some periodic communication according to the general peer-algorithm with an extension that the peers store some information about their state in a `GlobalView` object. The observer component will periodically check the status of the GlobalView and determine when the simulation can be terminated. Before termination the GlobalView is stored in a result-map which then is used by the test to conduct verdict-checks and assertions to verify the specific properties of the test. The assertions can be found in JUnit test-suites for each test in `/src/test/java/se/kth/tests/testname/sim/Testname.java`.

6.1 Gossiping Best-Effort-Broadcast

The component Gossiping Best-Effort-Broadcast (GBEB) provides three guarantees:

- *Validity* only when the sender is correct
- *No Duplication*
- *No Creation*

To test this component we have used the following scenario. N correct peers will send M distinct[3] broadcast-messages to each other. During the verdict-checks the test will verify that every peer (because they are correct) should have received all messages, i.e $N \cdot M$ messages.

6.2 Eager Reliable Broadcast

The component Eager Reliable Broadcast (EagerRB) provides an additional guarantee in extension to the guarantees of the GBEB component: the Reliable Broadcast property. The Reliable Broadcast property guarantees the Validity property even when the sender has failed. To test this component two scenarios were used.

I The first scenario will have two groups of peers, one group of core peers that are correct during the whole execution and a group of extension peers that are faulty and will be restarted[4] during the execution. At the end, the test asserts that all peers have received the same set of messages (all messages since all peers are eventually restarted).

II The second scenario uses the same idea as the first scenario with a set of core-nodes and a set of extension nodes. The scenario will first start all nodes and then kill of some of the extension nodes, wait a while and then restart the killed extension nodes. Finally, all of the extension nodes are killed. A difference compared to the first scenario is that when nodes are killed and restarted they are restarted as completely new nodes, effectively simulating churn. During its lifetime each peer will periodically send broadcasts. After the scenario is terminated, the test verifies that all *correct* nodes delivered the same set of messages. This test models the reliable-delivery property.

[3] intra and inter-peer thanks to random nonces

[4] Implementation detail: The restart of peers are simulated inside the HostMngrComp by destroying and recreating components.

6.3 No-Waiting Causal Broadcast

The last component among the broadcast abstractions is the No Waiting Causal-Order Broadcast (NoWaitingCB) that provide the property: Causal Order Delivery, which guarantees that messages are delivered in causal order, respecting the *happened-before* ($\rightarrow$) relation.

To test this component, a scenario in which N peers concurrently send M distinct broadcasts to each other appended with Lamport timestamps is simulated. When the simulation is terminated the test checks that all peers delivered the messages in causal order.

6.4 Logoot

The application component, AppComp maintains a Logoot-Undo document that is an implementation of the CRDT algorithm introduced in the paper [3]. Apart from unit tests on identifiers, patch and document, we have provided simulation-tests covering:

- Verifying that insertion and deletion operations commute
- Verifying that undo and redo operation work properly
- Tests of the Cemetery.

Insertion and Deletion operations tests For each operation we have tested that concurrent insertion and deletion operations were performed as they are supposed to by letting each peer periodically and concurrently do editions to the same document and then upon the termination of the scenario it is verified that each peer has the same document, i.e verified that all operations commuted.

Undo and Redo operations tests These tests aims to prove correctness of the operations as well as testing the degree of a patch introduced for this purpose. For each test, undo and redo operations are performed concurrently by the peers. However the undo and redo operations are delimited to operations previously made by the same peer performing the undo-operation. Upon the termination of the simulation the test will check that the resulting document is consistent to the patch executed.

Cemetery In the previous scenario the peers performed undo and redo operations concurrently but on distinct operations. This test aims to demonstrate the behavior of concurrent patches that considers the same operations. The scenario tested is based on the one introduced in the paper [3, Figure 6]. In this scenario, there are two sites with the same initial document, they both perform a patch that removes one line. Then, one site undo the other site's patch. The verdict-checks of the test will verify that the operations have commuted such that each peer has ended up with the correct document as shown in the paper-scenario.

Other In our implementation, we have not done tests on the performance of the Logoot algorithm, in the paper they have made their own tests on this point.

References

[1] Jim Dowling and Amir H. Payberah. Shuffling with a croupier: Nat-aware peer-sampling. In *ICDCS*, pages 102–111. IEEE Computer Society, 2012.

[2] Marc Shapiro and Nuno M. Preguiça. Designing a commutative replicated data type. *CoRR*, abs/0710.1784, 2007.

[3] Stephane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo: Distributed collaborative editing system on p2p networks. *IEEE Trans. Parallel Distrib. Syst.*, 21(8):1162–1174, August 2010.