Methods for Eventual Consistency in Collaborative Editing Systems

Methoden für eventuelle Konsistenz in kollaborativen Bearbeitungssystemen

Boyan Katsarski, Velko Iltschev

Department of Computer Science: Technical University Sofia, branch Plovdiv
Plovdiv, Bulgaria, Email: katzarsky@gmail.com, iltchev@tu-plovdiv.bg

***Abstract* —** The paper analyzes methods for Operational Transformations and models for Conflict-free Replicated Data Types in the light of Collaborative Editing Systems and preservation of semantics. Benefits and disadvantages of each method or model are presented. A proposal for semantic improvement and usability has been made, which includes: switching atoms from characters to words and adding inverse indexing to the documents with words stored private dictionaries.

***Zusammenfassung* —** Der Artikel analysiert Methoden für operative Transformationen und Modelle für konfliktfreie replizierte Datentypen in Hinblick auf kollaborativen Textverarbeitungssystemen und auf Erhaltung der Semantik. Vor- und Nachteile jeder Methode oder jedes Modells werden vorgestellt. Ein Vorschlag für semantische Verbesserung und Nutzbarkeit wird gemacht, der Folgendes umfasst: Anwendung ganzer Wörter anstelle einzelner Zeichen als Atome und Einfügen inverser Indexierung zu den Dokumenten mit den Wörtern, die in den privaten Wörterbüchern gespeichert sind.

I. Introduction

Users of mobile applications expect them to continue working even when the device is offline or has bad network connectivity and when the network becomes available again to synchronize with other devices. Such applications are: calendars, address-books, password-managers, task-organizers, etc. Likewise, collaborative work requires several users to edit the same text document (text, spreadsheet, etc.) at the same time. Each user's concurrent edits must be reflected on the others' replicas with minimal delay.

The key requirement here is that the state of the distributed application needs to be replicated on many devices (nodes). Each of them modifies the state locally and propagates the changes optimistically, while continuing to work locally.

The traditional approach to concurrency control - serializability (with locking), causes the application to become unusable when network connectivity is poor [1].

For applications, which tolerate temporal network outages, we must assume that users can make updates concurrently on different nodes and the resulting conflicts must be resolved.

The naive way to resolve conflicts is to discard some of the updates when a conflict arises. This approach leads to loss of updates and potentially data. Another approach is to make the user manually resolve the conflict using some logic or common-sense. This is tedious and prone to errors and should be avoided if possible. Nowadays applications solve this problem with a range of ad-hoc and application-specific means.

In this paper an overview of the benefits and disadvantages of the current general-purpose optimistic replication methods and models has been made. The focus is on Operational Transformation (OT) methods and Conflict-free Replicated Data Types (CRDTs) models and their derivatives in the context of collaborative editing systems. Finally, a proposal for semantic improvement and usability has been made, which includes: switching atoms from characters to words and adding inverse indexing to the documents with words stored private dictionaries.

II. Strong Eventual Consistency in Partitioned Networks. Collaboration.

A. Partitioned network

In the real networks, devices (especially mobile ones) go offline and then online all the time in a random manner, with periods of no connectivity in-between. Under these conditions it is impossible to achieve immediate strong consistency of the replicated state across all devices. Moreover, due to poor networks, updates made in the originating node may be received by the other nodes in a different order. While some of the updates are in transit and some of the replicas are only converging to the *eventual* consistent state.

B. Strong Eventual Consistency (SEC)

Eventual consistency requires that at the end – when all updates from all the nodes are sent and received – all replicated states will become the same.

More formally *eventual consistency* is defined [2] as:

- **Eventual Delivery**: Every node will see the updates made by the other nodes *eventually*. These updates may come out-of-order due to network partitioning.
- **Convergence:** Same operations should lead to the same state. Even when received out-of-order.
- **No updates are lost due to concurrent modifications:** A good example for a violation is the "Last Writer Wins" (LWW) policy.

C. Collaborative Editing Systems (CES)

In the constraints imposed by partitioned networks and eventual consistency, collaborative editing becomes a non-

trivial task.

Nowadays, collaborative editing systems are based on the old Operational Transformation principles. There is however a new approach – CRDT, which is still in research and development.

III. Approaches to Optimistic Replication, Collaborative Editing and Conflict Resolution

A. Operational Transformation (OT)

The algorithms for Operational Transformation appear first in the works of Ellis & Gibbs [3] in 1989. Later improvements are made by M. Ressel et al [4] and C. Sun et al [5]. Most of them treat a document as a single list of characters.

First OT system is GROVE based on the "dOPT" algorithm [3]. Currently Google Docs uses a modified version of it.

The essence of OT is that operations made in one node are *transformed* on the fly when applied to another node in order to get the same resulting state. A transformation function is used to change the position of the incoming operations in order to be adequately applied to the local text.

For example: if Alice deletes a character at position 5 while concurrently Bob inserts a character at position 10. Bob's operation should be transformed to position 9 = (10-1) when applied to Alice's text.

The transformation function must take into account all prior operations made locally. Moreover, if there are many concurrent editors, the transformations become exponentially more complex as the operations come in different order for each of the users. Complexity is aggravated when an *offline* user has produced a big batch of operations and now becomes *online*. This makes OT unsuitable for many users working in poor networks.

Notable implementations are Grove (1989), Jupiter (1995) and its descendants: Google Docs, Apache Wave (formerly Google Wave), Etherpad and ShareJs.

To alleviate the problem with the complexity mentioned above Google Docs applies restrictions like:

- Using a single centralized server to sequence the order of updates.
- The centralized server makes some or all (details are proprietary) of the transformations of the operations.
- Restricted or disabled offline editing capabilities to limit the transformations produced by stale updates. (Stale updates lead to recalculation of all the updates made in the meantime.)

There are a couple of problems with these limitations:

- A centralized server introduces a single point of failure.
- A centralized server makes transformations and it needs to have the operations' content, which means end-to-end encryption is impossible.
- Offline work is limited severely.

B. Conflict-free Replicated Data Types (CRDT)

Later in 2011 a notion for special replication-ready data model appears in the works of Shapiro et al [6].

CRDT considers that transforming operation is too complex and establishes a new model to handle real-time collaboration. While OT attempts to make non-commuting operations commute after the fact (via transformation). A better approach is to design operations to commute in the first place. This avoids the complexities of OT.

Operations that update the model must follow 3 constraints:

- Commutative
- Associative: these two ensure that operations can be applied in any order
- Idempotent: an operation can be applied many times and it yields the same result as if applied once.

The first two (commutative & associative) deal with the out-of-order problem due to poor networks. The last one (idempotent) remedies the problem that arises when an operation is sent to a remote node and no acknowledgement is returned. At this point it is unclear whether the node has *not received* the operation or it has received it but *not acknowledged* it or the acknowledgement has been *lost* in the network. Hence, CRDT operations need idempotency.

Model with these restrictions makes conflicts impossible [6], but as a result, the model is monotonically growing (while leaving tombstones) and designing operations with the above constraints becomes difficult quickly.

The essence of CRDT is that operations should be designed following the 3 rules above. A trivial example is an increment-only counter, which in CRDT terms must be designed as an array of integers [6]. Each index is owned only by one user. Write operations are done only to the user's element but incoming updates can be received on all elements of the array. The actual value of the counter is the sum of all elements.

The benefits of the CRDT model include:

- Built-in strong eventual consistency
- Possible server-less implementation, providing end-to-end encryption and no single point of failure
- Offline capability with no extra complexity.

The disadvantages are:

- Ever-growing state due to more users added or data being "deleted" under tombstones [7]. Distributed garbage collection methods are non-trivial and require some sort of locking.
- Difficult to design operations (API) for the CRDT type (map, set, counter, text document, etc.)

Some of the trivial CRDTs include [8]:

- Increment-Only Counter
- PN (increment/decrement) Counter
- Add-Only Set
- Directed Graph CRDT
- Last-Writer-Wins Register

Notable implementations of trivial CRDTs are made by: Riak (library), Bet365 (counters) and League of Legends (chat).

Some of the non-trivial CRDTs (plus algorithms) for collaborative text editing are:

- **WOOT** by Oster et al [9]
- **Treedoc** by Nuno Preguiça et al [10]
- **RGA** by Roh et al [11]
- **Logoot** by Weiss et al [12]

They share some common characteristics. A character is considered an *atom* in the model (recently a UTF-8 character which is represented by more than one byte). The structure varies (linked lists, trees and semi-lattices) but to *preserve the intent* of the original editor tombstones [7] are used. Tombstone is an atom marked as "deleted" and skipped when presenting to the user. Tombstones are needed when an atom is deleted but it serves as a reference point for concurrent insert.

IV. CRDTs for Collaborative Editing

A. WOOT

The "WithOut Operational Transformation" method is developed by Oster et al [9] in 2005. The essence of it is that it treats characters as atoms in a *linked list.* Each element has id indicating precedence. Inserts contain <character, id, preceding-id>. Deletes just mark the id as a tombstone. There is no update operation but delete and insert. Tombstones are needed when one user deletes a character while another concurrently adds a character after it. Generation of IDs is special (free from vector clocks) so that a sorting function can linearize the resulting semi-lattice. A filtering function skips the tombstones when presenting the content to the user.

The method works well even with stale updates and does not explode in complexity when more users join the editing, unlike any OT method.

Operations Complexities [11]:

- Local: Insert $O(N^2)$, Delete O(1)
- Remote: Insert $O(N^3)$, Delete O(N)

Main problem is the unbound growth of the document when lots of edits are submitted over time. The garbage collection of the tombstones is non-trivial and requires all nodes to be online to reach a consensus on which tombstones are not needed anymore and will be physically deleted from the document. This is aggravated by the fact that the atom is a character and update of a character is translated to delete and insert, producing a tombstone in the process. Algorithm of purging the tombstones is not yet presented.

B. Treedoc

Treedoc is presented by Nuno Preguiça et al [10] in 2009. It stores atoms in a balanced binary-tree structure (with the extension of *mini-nodes* to handle more than two children stemming from the same parent). Treedoc is a binary tree whose paths to nodes are unique indices and ordered totally in infix order. It tries to keep the tree balanced and executes *explode* and *flatten* routines over a subtree to minimize tombstones and tree structure overhead. Flattening uses voting commit, similar to ACID databases' two-phase commit. Still uses tombstones.

It works but has issues:

- if the user appends at the end of the document it unbalances the tree
- optimizations for tree flattening require locking which makes it less usable in poor networks
- a lesser problem is the overhead of maintaining the tree structure balanced

C. RGA

"Replicated Growable Array" is proposed by Roh et al [11] in 2011 as a Replicated Abstract Data Type (RADT) aiming for text and data modelling. It treats the text as a linked-list of linked-lists (text blocks). It uses hash-tables to accelerate the access and improves garbage collection of tombstones by adding them to a *cemetery*.

Operations Complexities [11]:

- Local: O(N)
- Remote: O(1)

Complexity is lower than earlier models due to hashes. Moving text fragments (copy/paste) is simpler because a text-block is represented by a sub-linked-list. It uses tombstones to preserve intention as well.

Negatives include the overhead of supporting tombstones and the problem with inserting new text at the same coordinates concurrently.

D. LOGOOT

A CRDT proposed by Weiss [12] in 2009. A Logoot document is composed by lines defined by: <*id, content*> where *content* is a text line and *id* - a unique position identifier. Its structure supports easy deletion and insertion of new line(s). It does NOT rely on tombstones for consistency. Complexity is logarithmic. It preserves the meaning of a text line better because a user can only add/remove a line, not a single character. Atom ownership is not over a character but a line.

V. Common Problems Plaguing the CRDTs for Collaborative Editing

A. Atom Ownership is over characters

Most of the CRDT for text editing, *except Logoot* are based on atoms that are characters. This is a problem because meaning is contained not within a single character but rather within words, phrases and sentences. Assigning blame for badly formed phrases that result in a collaborative editing is difficult since one word may be edited by couple of users. Only Logoot hits the mark in that area with having lines of text as atoms, which is not ideal but at least addresses the issue. A better solution may be using words or phrases as atoms.

B. Moving Text Blocks

Moving big blocks of text is a common practice when editing. Most of the methods above *except Logoot* suffer from structural issues when this is done: unbalanced trees and massive tombstone production. Since current tombstone garbage collection algorithms require locking commits they compromise the P2P-ready nature of CRDTs. A better approach for tombstone purging is needed or a method that does not even require them.

C. Search

Yielding the resulting text in CRDTs that are based on characters and tombstones is another issue because the text is calculated rather than used "as-is" and searching through documents based on these models require an extra step before the actual search.

D. Concurrently Inserting at the Same Point

It is "solved" in current models by inserting both fragments while *stabilizing* the order of the inserts. But this approach will produce semantically questionable results and is still unsolved semantically by any of the models.

VI. Conclusion. And a Way Forward

A step towards a more meaningful policy of semantic ownership is assigning words or phrases as atoms. Since phrases are difficult to isolate even using grammatical analysis for different natural languages – words will be a wiser choice. An editor can be built on top to color-code the ownership of words, hence assigning blame for bad phrasing can be done easily.

Isolating words that are capitalized or punctuated may be represented as the non-punctuated word with added attributes to the atom. This will keep the dictionary smaller and more meaningful, since the words will be in cleaner form.

These word-atoms can form a dictionary that will enable inverse document storage right from the start. The inverse indexing will enable *built-in search* of the edited text. The document structure will use dictionary-pointers instead of the words themselves.

The dictionary can be partitioned in sections owned by the users. Since a section is owned only by one user – it can be

garbage collected only by the owner-user and not requiring a consensus or a lock protocol.

The editor which can be built on top of this proposed data structure will have to observe user actions such as:

1. typing or deleting a character
2. copy-pasting or moving a block of text
3. deleting a block of text

Adding or deleting a character (1) in a word will introduce temporary noise in the dictionary since the currently edited word will change. But this will affect only the private section of the user-dictionary. If the word was not owned by the editing user the new word is created in the user-owned partition of the dictionary.

Copy-pasting or cut-pasting (2) a big block of text will not make a lot of noise. The dictionary will stay mostly unchanged, except in the case when the text block cuts words in the middle. The structure made of dictionary-pointer atoms will reflect the change as in WOOT or LOGOOT while keeping the atoms count low.

Deleting a block of text (3) will leave fewer tombstones, since they are word-pointers not characters.

References

[1] S. B. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in partitioned networks," *ACM Computing Surveys*, vol. 17, no. 3, pp. 341–370, Sep. 1985.

[2] Kleppmann, Martin & R. Beresford, Alastair. (2016). "A Conflict-Free Replicated JSON Datatype." *IEEE Transactions on Parallel and Distributed Systems*. Volume 28, Issue 10, Oct 2017, pp. 2733-2746. 10.1109/TPDS.2017.2697382.

[3] C. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," in *ACM International Conference on Management of Data* (SIGMOD), May 1989, pp. 399-407.

[4] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauer, "An integrating, transformation-oriented approach to concurrency control and undo in group editors," in *ACM Conference on Computer Supported Cooperative Work* (CSCW), Nov. 1996, pp. 288-297.

[5] C. Sun and C. Ellis, "Operational transformation in real-time group editors: Issues, algorithms, and achievements," in *ACM Conference on Computer Supported Cooperative Work* (CSCW), Nov. 1998, pp. 59-68.

[6] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of convergent and commutative replicated data types," *[Research Report] RR-7506, INRIA*. 2011, pp.50.

[7] G. Oster, P. Urso, P. Molli, and A. Imine, "Tombstone transformation functions for ensuring consistency in collaborative editing systems," *IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing* - CollaborateCom 2006, Nov 2006, Atlanta, Georgia, USA, pp.1-10. 10.1109/COLCOM.2006.361867

[8] Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski. "Conflict-free Replicated Data Types.", *Stabilization, Safety, and Security of Distributed Systems*, Springer Berlin Heidelberg 2011, pp. 386-400.

[9] Gérald Oster, Pascal Urso, Pascal Molli, Abdessamad Imine. "Real time group editors without Operational transformation", *[Research Report] RR-5580, INRIA*. 2005, pp. 24.

[10] Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, Mihai Leția. "A commutative replicated data type for cooperative editing". *29th IEEE International Conference on Distributed Computing Systems* (ICDCS 2009), Jun 2009, Montreal, Québec, Canada. IEEE Computer Society, pp. 395-403. 10.1109/ICDCS.2009.20.

[11] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, Joonwon Lee. "Replicated abstract data types: Building blocks for collaborative applications", *Journal of Parallel and Distributed Computing*, Volume 71, Issue 3, March 2011, pp. 354-368.

[12] Stéphane Weiss, Pascal Urso, Pascal Molli. "Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks". *29th IEEE International Conference on Distributed Computing Systems* - ICDCS 2009, Jun 2009, Montreal, Canada. IEEE, pp. 404-412.