CRATE: Writing Stories Together with our Browsers

Brice Nédelec
Université de Nantes, LINA
2 rue de la Houssinière
Nantes, France

Pascal Molli
Université de Nantes, LINA
2 rue de la Houssinière
Nantes, France
first.last@univ-nantes.fr

Achour Mostefaoui
Université de Nantes, LINA
2 rue de la Houssinière
Nantes, France

ABSTRACT

Real-time collaborative editors are common tools for distributing work across space, time, and organizations. Unfortunately, mainstream editors such as Google Docs rely on central servers and raise privacy and scalability issues. CRATE is a real-time decentralized collaborative editor that runs directly in web browsers thanks to WebRTC. Compared to state-of-the-art, CRATE is the first real-time editor that only requires browsers in order to support collaborative editing and to transparently handle from small to large groups of users. Consequently, CRATE can also be used in massive online lectures, TV shows or large conferences to allow users to share their notes. CRATE's properties rely on two scientific results: (i) a replicated sequence structure with sub-linear upper bound on space complexity; this prevents the editor from running costly distributed garbage collectors, (ii) an adaptive peer sampling protocol; this prevent the editor from oversizing routing tables, hence from letting small networks pay the price of large networks. This paper describes CRATE, its properties and its usage.

Keywords

Collaborative editor; decentralized; real-time

1. INTRODUCTION

Real-time collaborative editors allow authors to simultaneously write shared documents. Trending editors such as Google Docs rely on central servers. They raise privacy issues since service providers take advantage of their mediating position to observe all users and documents. They also raise scalability issues, for they require a server to support an editing session, so handling many editing sessions requires many resources. Managing large editing sessions puts a lot of pressure on the server which may become overloaded.

CRATE is a distributed and decentralized collaborative editor providing scalable real-time editing capabilities directly within web browsers. Compared to state-of-the-art, CRATE is the first real-time editor that only requires browsers in order to support collaborative editing and to transparently handle from small to large groups of users.

Suppose massive online lecture platforms allow students to share their notes. Many lectures run in parallel involving various number of students, i.e., from few to thousands. Also, even during the editing session the audience is subject to significant changes in size, going from thousands to hundreds. Finally, the resulting document size is hardly predictable, for it depends of the number and zealous of involved students.

CRATE handles these situations by combining different algorithms to achieve an acceptable trade-off between space, time and communication complexities. In laboratory, we observed this trade-off on configurations involving up till 600 browsers. In this paper, we describe the usage, the architecture and the properties of CRATE. Also we propose a live experiment to WWW2016 participants aiming to confirm laboratory experiments about CRATE.

Section 2 depicts the usage of CRATE. Section 3 details the overall architecture of CRATE along with its basic functioning. Section 4 shows the way to build decentralized networks directly in browsers. Section 5 describes the distributed data structure that represents the document. Section 6 shows our laboratory result and describes the live experiment setup. Section 7 concludes.

2. USAGE

CRATE's video, source code and online demo are freely available at `https://github.com/Chat-Wane/CRATE`.

In the online demo[1], a user creates a document by clicking the $\oplus$ icon at the top of the screen (see Figure 1). At the time, the document is only local and no one can read nor modify it apart from its creator. When the user is ready to share, she clicks on the *chain* icon and CRATE is crafting an editing session URL that can be sent by mail, published on Twitter, or advertised by any other mean.

Once the collaborators open the link, it automatically connects them to the editing session of the creator. It retrieves the shared document and they can start the real-time collaborative editing. In turns, they are able to share the access to the document too.

Figure 1 shows a screenshot of the graphical user interface of CRATE. In this scenario, two editors are running in a same instance of the web application. The editing session comprises two members nonetheless. The leftmost author

[1] `http://chat-wane.github.io/CRATE/`, the browser must support WebRTC

WWW'16 Companion, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4144-8/16/04.
DOI: http://dx.doi.org/10.1145/2872518.2890539

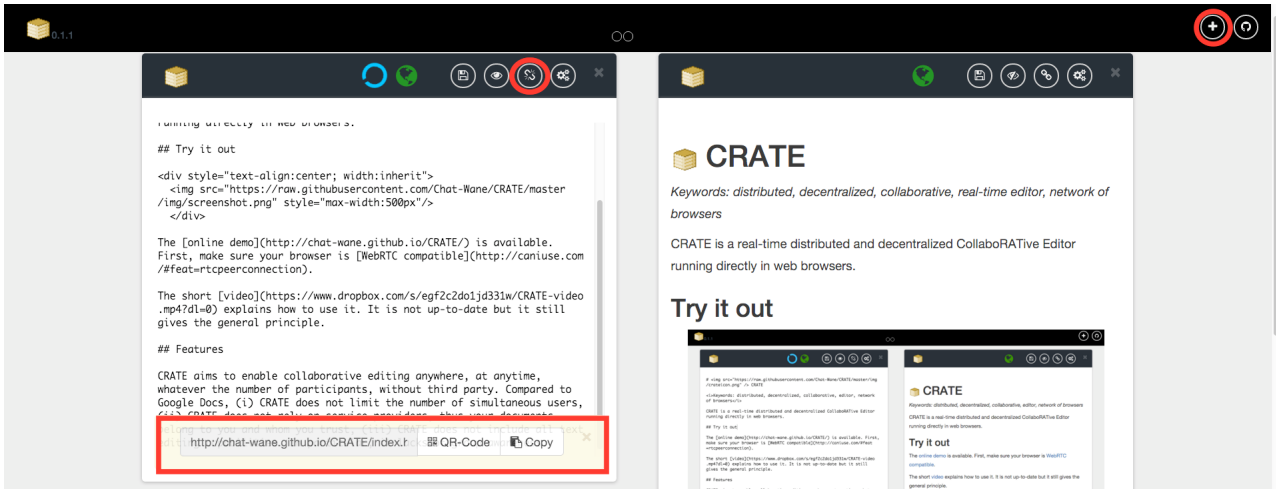

Figure 1: Screenshot of the web application containing two connected editors: on the left, a document is written in markdown language which is previewed on the right editor.

writes the *readme* file of the Github repository about CRATE. The green earth indicates that the editing session is live. The blue (spinning) circle indicates that she opens the access to her document. Therefore, anyone can join it with the proper URL. The rightmost editor renders the document written in markdown language. It has a green earth indicating a connected state but contrarily to the leftmost editor, it is not sharing the access to the document.

URLs are the means for accessing to editing sessions, hence, to documents. Internally, when a browser opens the link it retrieves the web application. The latter queries a mediator server with the parameters contained within the link (see Figure 3). The mediator server tries to establish the very first contact between the joining browser and an editing session sharer. The mediator ensures solely a minimal service consisting in the editing session accessibility.

Yet, an URL does not always grant the access to the live editing session, for there must be at least one sharer among the editors. Also, there may exist multiple URLs leading to a same editing session using different joining parameters. As we observe in Figure 1, CRATE enables hyperlinks. By the same, it allows surfing between editing sessions like any user would do on Wikipedia, enhanced with real-time capabilities.

3. ARCHITECTURE

To provide availability and responsiveness of documents, CRATE follows the optimistic replication scheme. Each editor replicates the document locally and directly performs operations on it. Next, the editor spreads its changes to all other participants. The system is correct iff editors integrating a same set of changes have replicas converging to an equivalent state, i.e., users read the same document. This property is called strong eventual consistency [4].

CRATE uses a Conflict-free Replicated Data Type (CRDT) for sequences to ensure such property [4]. CRDTs ensure consistency at the price of a unique identifier attached to each element of the sequence. Recently, LSEQ [2] proposed a strategy to bound the space complexity of these identifiers to $\mathcal{O}(\log(d)^2)$ where d is the document size. As such, CRATE does not require costly distributed garbage-collection-like mechanisms to maintain its efficiency.

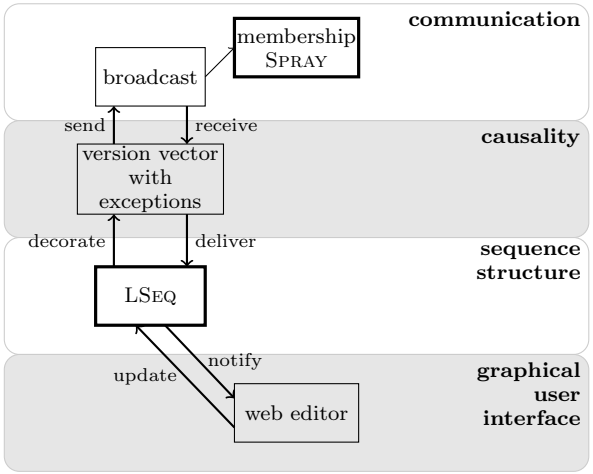

Figure 2: Four layers architecture of CRATE.

Strong eventual consistency requires the eventual delivery of all operations to all editing session members. CRATE builds an editing session using SPRAY [3], a random peer sampling protocol [1] (RPS) built on top of WebRTC. A peer sampling protocol provides each network member with a partial view of the network significantly smaller compared to this latter. Maintaining randomness among partial views ensures the connectivity of the network. Unlike prior RPS, SPRAY allows adapting the partial view sizes to the editing session size without measuring its number of participants. Since the propagation protocol of messages extensively uses these tables, the network traffic inherits this scalability. CRATE adapts its operation to the needs of the editing session.

The decentralized collaborative editor CRATE is developed in web languages only: HTML, CSS, and mostly JavaScript. As shown by Figure 2, CRATE comprises four layers:

The graphical user interface that renders the document to the users (cf. Figure 1). Each local update is immediately applied to the document view for real-time sake. Then, the update is applied to the replicated data type;

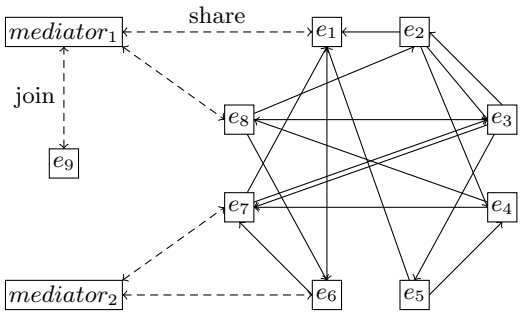

Figure 3: Example of a small network built by SPRAY. Each editor e can communicate with a small partial view of the network. For instance, Editor e_8's partial view comprises e_2, e_3 and e_6. In addition, e_1 and e_8 grant a public access to the editing session through $mediator_1$. Editor e_9 is joining.

The sequence structure layer that represents the local document (cf. §5). It is in charge of providing the metadata necessary to order each character identically anywhere;

The causality layer that tracks semantic causality between operations, e.g., the removal of a character cannot precede its insertion;

The network layer that (i) builds a network of browsers for each editing session and (ii) uses it to propagate the updates to all collaborators (cf. §4).

4. NETWORK OF BROWSERS

CRATE uses SPRAY [3] to build a network of browsers thanks to WebRTC. The latter is a recent technology that allows establishing browser-to-browser channels of communication. Its connection establishment protocol imposes a round-trip of messages. Thus, if an editor e_1 wants to connect to another editor e_2, it needs to send a message to e_2 and to acknowledge its response. While the first of such connection must use a mediator (e.g. mail, or dedicated server), once established, the network can assume the position of mediator and start working autonomously.

Random peer sampling protocols provide each editor with a local neighborhood table. These tables allow communicating to a subset of editors. Messages travel from neighbor to neighbor to reach any editor in a scalable manner.

SPRAY's lifecycle at each editor comprises three steps that aim to provide and maintain logarithmically scaling neighborhood tables compared to the editing session size:

The joining: the editor wants to join the editing session. It contacts an editor inside the network and waits for its answer to establish the very first WebRTC connection between them. Using this connection, the joining editor asks to its contact editor to operate as mediator to advertise its presence to the contact's neighbors. Each of these neighbors connects to the joining peer. Since the number of neighbors is supposed logarithmic, the number of connections in the network grows by $\log(N)+1$ where $\log(N)$ is the natural logarithm of the network size N. Following such growth, the global number of connections is $N.\log(N)$.

The shuffling: each editor regularly initiates a table shuffling with the oldest of its neighbors. The initiator serves as mediator between half of its neighborhood chosen at random and the oldest neighbor. In response, the oldest neighbor does the same. As a result, they roughly have an identical neighborhood size which tackles the load-balancing problem that may rise from the joining part of the protocol. It is worth noting that the number of connection remains constant. In addition, the references in neighborhood tables are randomly distributed. The network converges in exponential time toward a topology exposing properties similar to those of random graphs (cf. Figure 3).

The leaving: the editor can leave the network without giving notice, i.e., equivalent to a crash. Nevertheless, its removal from the system must lead to a decrease of $\log(N)+1$ connections while currently it represents $\log(N)$ connections from its neighborhood tables, and $\log(N)$ connections from other editors' neighborhood tables pointing to it. The adjusting falls under the responsibility of these other editors. For this purpose, they detect the departure of a neighbor when they try to initiate a shuffle with it. Since such detections presumably happen $\log(N)$ times, they all duplicate one of their neighbors except for one; the probability being processed using their own neighborhood table size.

Random peer sampling protocol constitutes the ground of various protocols such as topology managements. CRATE uses it to disseminate all operations of the replicated sequence to all collaborators in a scalable manner. Referred as epidemic dissemination, rumor mongering, or gossiping, the protocol is a two-step operation: (i) Each operation performed on the local document replica leads to the creation of a message. The latter is sent to all the neighborhood provided by SPRAY. (ii) Each editor receiving such broadcast message transmits it again to its neighborhood. Thus, messages transitively reach all collaborators very efficiently. This dissemination protocol inherits from SPRAY's adaptiveness. Hence, traffic follows the editing session size.

Prior approaches generally oversize the partial views to handle largest groups of users. Unfortunately, smaller groups suffer from such *a priori* dimensioning. Since SPRAY automatically adjusts neighborhood table sizes, it equally and transparantly handles from smallest to largest groups.

5. REPLICATED DOCUMENT

To provide eventually consistent replicas, CRATE uses a conflict-free replicated data type for sequences [4]. Such sequence type avoids the difficult and error-prone task of solving conflicts by the mean of commutative operations. Thus, the sequence type provides two basic operations: the insertion and removal of an element. If two users perform concurrently some operations at an identical position in the sequence, the latter will still converge to an equivalent state.

Commutativity comes at the price of additional metadata attached to each element of the sequence. These metadata referred to as identifier are unique, immutable, and of variable size at generation. When a character is inserted, the sequence type generates an identifier which is a list $[\ell_1.\ell_2 \dots \ell_k]$ where k is the identifier depth and Element ℓ_i comprises a path p_i, a globally unique site identifier s_i, a monotonic local counter c_i.

A lexicographic total order amongst these identifiers allows retrieving the sequence:

$$(1)\ \ell_i < \ell_j \iff (p_i < p_j) \vee ((p_i = p_j) \wedge (s_i < s_j)) \vee ((p_i = p_j) \wedge (s_i = s_j) \wedge (c_i < c_j))$$

$$(2)\ \ell_i = \ell_j \iff \neg(\ell_i < \ell_j) \wedge \neg(\ell_j < \ell_i)$$

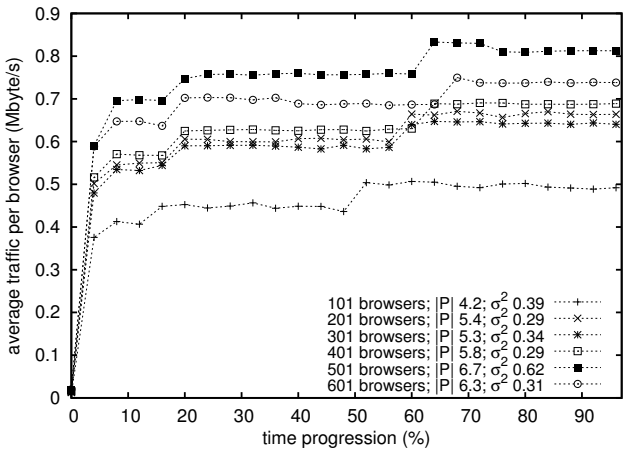

Figure 4: Average traffic per second.

(3) $i_i < i_j \iff \exists(m > 0)(\forall n < m),\ (\ell^i_n = \ell^j_n) \wedge (\ell^i_m < \ell^j_m)$

(4) $i_i = i_j \iff \forall m,\ \ell^i_m = \ell^j_m$

The first two equivalences define an ordering between elements of the list composing the identifiers. The last two equivalences use them to define a global total order amongst identifiers, hence, ordering characters of the document. While the third equivalence is mostly used to locate the proper position of insertion, the fourth equivalence is necessary for removals. Paths of identifiers constitute the most discriminant part of identifiers, hence the most important part.

CRATE uses LSEQ to generate the paths of identifiers. The paths are series of integers $[p_1.p_2 \dots p_k]$ where each path is chosen among a set twice as large as its preceding path. For instance, if p_1 is chosen among $\{0..2^8\}$ then p_2 is chosen among $\{0..2^9\}$ etc. When the document grows, the depth of identifiers is expected to grow. Therefore, increasing the size of sets over depths decreases the growth in depth. LSEQ allocates identifiers polylogarithmically upper-bounded compared to the number of insertions, without *a priori* knowledge of the series of updates.

Keeping the size under such a sub-linear bound avoids the need of an unafordable distributed garbage-collecting-like mechanism.

6. EXPERIMENT AND DEMONSTRATION

In laboratory, we tested CRATE on the Grid'5000 testbed with configurations involving up till 600 browsers. Figure 4 shows the traffic generated at each member by intensive editing sessions. Overall, the artificial authors insert 100 characters per second during 7 hours. The documents reach millions of characters. Figure 4 shows the combined effects of SPRAY and LSEQ. Indeed, we observe that the traffic logarithmically scales to the editing session size thanks to SPRAY. The members of the smallest editing session (101 browsers) are less traffic intensive than ones of the largest group (601 browsers). But the messages transiting the network are important too. The growth of each plot corresponds to the identifiers size generated by LSEQ. Since the document size grows, the identifiers grow, but their growth slows over insertions.

We would like to confirm the results of the experimentation by performing a live demonstration of CRATE that any WWW2016 participant can join. We will start an *exquisite corpse*[2] collaborative storytelling. In this game, we will invite every participant to continue or update the story initiated by previous participants.

In that regard, we will bootstrap an initial document in our local browser and share it through a public URL advertised on Twitter. Every participant will be able to join the session by just clicking on this link. Next, she will freely add a sentence. During the editing session, we will invite participants to share and advertise the document with their friends in order to get as many participants as possible.

During the experiment, we will be able to monitor the evolution of the document and network. We expect the space complexity of the identifiers associated to each character to be upper-bounded by $\log(d)^2$ where d is the number of characters. We expect the partial view sizes to stay at $\log(N)$ where N is the number of participants at a given time.

7. CONCLUSION

In this paper we described CRATE, a distributed and decentralized real-time collaborative editor running in web browsers. Compared to state-of-the-art, CRATE allows real-time editing anytime, anywhere, whatever the number of participants. It can be seen as an alternative way to provide a collaborative editing service without Cloud support. Using a scalable replicated data structure for sequences, and an adaptive peer sampling protocol, CRATE is able to alleviate scalability issues and privacy issues without compromising ease-of-access. CRATE's video, source code and online demo are available at `https://github.com/Chat-Wane/CRATE`.

8. ACKNOWLEDGMENTS

This work was partially funded by the French ANR project SocioPlug (ANR-13-INFR-0003), and by the DeSceNt project granted by the Labex CominLabs excellence laboratory (ANR-10-LABX-07-01).

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

9. REFERENCES

[1] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)*, 25(3):8, 2007.

[2] B. Nédelec, P. Molli, A. Mostéfaoui, and E. Desmontils. LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing. In ACM, editor, *13th ACM Symposium on Document Engineering*, Sept. 2013.

[3] B. Nédelec, J. Tanke, D. Frey, P. Molli, and A. Mostefaoui. Spray: an Adaptive Random Peer Sampling Protocol. Technical report, LINA-University of Nantes ; INRIA Rennes - Bretagne Atlantique, Sept. 2015.

[4] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Rapport de recherche RR-7506, INRIA, Jan. 2011.

[2] `https://en.wikipedia.org/wiki/Exquisite_corpse`