LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing

Brice Nédelec, Pascal Molli, Achour Mostefaoui, Emmanuel Desmontils

LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing

Brice Nedelec, Pascal Molli, Achour Mostefaoui, Emmanuel Desmontils
LINA, 2 rue de la Houssinière
BP92208, 44322 Nantes Cedex 03
first.last@univ-nantes.fr

ABSTRACT

Distributed collaborative editing systems allow users to work distributed in time, space and across organizations. Trending distributed collaborative editors such as Google Docs, Etherpad or Git have grown in popularity over the years. A new kind of distributed editors based on a family of distributed data structure replicated on several sites called Conflict-free Replicated Data Type (CRDT for short) appeared recently. This paper considers a CRDT that represents a distributed sequence of basic elements that can be lines, words or characters (sequence CRDT). The possible operations on this sequence are the insertion and the deletion of elements. Compared to the state of the art, this approach is more decentralized and better scales in terms of the number of participants. However, its space complexity is linear with respect to the total number of inserts and the insertion points in the document. This makes the overall performance of such editors dependent on the editing behaviour of users. This paper proposes and models LSEQ, an adaptive allocation strategy for a sequence CRDT. LSEQ achieves in the average a sub-linear spatial-complexity whatever is the editing behaviour. A series of experiments validates LSEQ showing that it outperforms existing approaches.

Categories and Subject Descriptors

I.7.1 [**Document and Text Processing**]: Document and Text Editing—*Document management*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.2.8 [**Software Engineering**]: Metrics—*Complexity measures*

Keywords

Distributed Documents; Document Authoring Tools and Systems; Distributed Collaborative Editing; Real-time Editing; Conflict-free Replicated Data Types

DocEng'13, September 10–13, 2013, Florence, Italy.

http://dx.doi.org/10.1145/2494266.2494278.

1. INTRODUCTION

Distributed collaborative editing systems [4, 5, 6] such as Google Docs, Etherpad or Git are now widely used and allow users to work distributed in time, space, and across organizations. A new kind of distributed editors [12, 20] appeared based on Conflict-Free Replicated Data Types (CRDTs) [11, 21, 16]. A CRDT is a distributed data type replicated over several sites [15, 14]. A CRDT cannot implement any centralized data structure but for instance can implement a counter, a set, a tree, etc. In this paper we consider a special family or CRDTs that implement a sequence of basic elements such as lines, words or characters that we call *sequence CRDT*. For our purpose and as a first step, we only consider two basic operations on a sequence, the insert and the delete operations. Compared to the state of the art, editors based on sequence CRDTs are more decentralized and scale better. However, they have a linear space-complexity with respect to the number of insertions. Consequently, they heavily depend on the editing behaviour. Some editing scenarios lead to a permanent loss in performance.

In order to preserve the total order on the elements of the sequence, a unique and immutable identifier is associated with each basic element of the structure (character, line or paragraph according to the chosen granularity). This allows distinguishing two classes of sequence CRDTs: (i) Fixed size identifier (also called the tombstones class). This class includes WOOT [11], WOOTO [20], WOOTH [1], CT [7], RGA [13], [23]. In this class, a tombstone replaces each suppressed element. Although it enjoys a fixed length for identifiers and it has a space complexity which depends on the number of operations. For example, a document with an history of a million operations and finally containing a single line can have as much as 499999 tombstones. Garbaging tombstones requires costly protocols in decentralized distributed systems. (ii) Variable-size identifiers. This class includes for example Logoot [21]. It does not require tombstones, but its identifiers can grow unbounded. Consequently, although it does not require garbage protocols, its space complexity remains till now linear with the number of insert operations. Thus, it is possible to have only a single element in the sequence having an identifier of length 499999. Treedoc [12] uses both tombstones and variable size identifiers but relies on a complex garbage protocol when identifiers grow too much.

In this paper, we propose a new approach, called LSEQ, that belongs to the variable-size identifiers class of sequence CRDTs. Compared to the state of the art, LSEQ is an adaptive allocation strategy with a sub-linear upper-bound

in its spatial complexity. We experimented LSEQ on synthetic sequences and real documents. In both cases, LSEQ outperforms existing approaches.

The remainder of the paper is organized as follows. Section 2 delineates the background on variable-size identifiers class of sequence CRDTs and motivates this work. Then, Section 3 details LSEQ and the three parameters that control the growth and the selection of the identifiers. It describes each parameter with its aim and its defect. Also, how their composition overcomes their respective weaknesses. Section 4 validates the approach by showing the effect of each one of the three parameters of LSEQ and also by comparing the proposed approach to Logoot. Finally, Section 5 reviews related works.

2. PRELIMINARIES

Distributed collaborative editing systems consider a sequence of characters replicated on n sites. Each site manages a local copy of the sequence called local replica. At some moments, the local replica can differ from some other sites' local replica. Each site can *insert* and *delete* characters in the sequence without any locking mechanism. Then, all sites exchange and deliver the operations. When any site delivers an insert operation, the state of the local replica may be different from the state of another replica.

The system is correct if: (i) It converges i.e. all local replicas of the sequence are equal when the system is idle. It corresponds to the eventual consistency property [8]. (ii) It preserves all partial order relations $\prec$ between characters. If a site inserts the character x between the characters a and b ($a \prec x \prec b$), this relation is preserved on each sites' replica. It corresponds to the property of intention preservation in Operational Transformation algorithms [19] used by Google Docs.

Let us illustrate this with an example. Assume that all the replicas of a sequence of characters are equal and that the sequence looks like *...abcd....* Consider that a site inserts the character e between b and c and also consider that at the same moment, another site inserts a character f between b and c. It results in two relations $b \prec e \prec c$ and $b \prec f \prec c$. Once every site has delivered all the changes on their local replica, the union of these two relations merges into a partial relation without any precedence between e and f. Consequently, two final states are possible $abefcd$ and $abfecd$. The role of the sequence CRDTs is to build a linear extension of the partial order formed by the intentions of all users to obtain a unique total order.

Variable-size sequence CRDTs encode order relations into identifiers. For example, the operation $insert(a = 10 \prec x \prec b = 15)$ can be sent as $insert(x, 12)$. This strategy does not require keeping tombstones, however it is easy to see that identifiers can grow quickly and significantly degrade the overall performance of the system. In the worst case, the system requires to re-balance identifiers implying the use of a consensus algorithm.

In this paper, we focus on keeping the identifiers as small as possible hence avoiding any costly protocol to re-balance them. Definition 1 states a document as a set of pairs (elt, id) where elt can be a character or a line and id are unique immutable identifiers defined on the set of all possible identifiers $\mathcal{I}$. $\mathcal{I}$ has an order relation $<$ which is dense and strictly totally ordered i.e. if $x, y \in \mathcal{I}$ and $x < y$ then $\exists z \in \mathcal{I}, z \neq x, z \neq y, x < z < y$. $alloc(p, q)$ is the allocation strategy function that generates $id \in \mathcal{I}$. In Definition 2, we state that an id is a sequence of numbers, $id_1 < id_2$ if id_1 precedes id_2 in lexicographic order. This sequence is an efficient way to represent a dense order.

DEFINITION 1 (MODEL OF A DOCUMENT). *A document is a set $\mathcal{D} = \{(elt, id)\}$ with two operations:*

— *insert($p \in \mathcal{I}$, elt, $q \in \mathcal{I}$):-* $\mathcal{D} \cup \{(elt, id_{elt})\}$ *where* $id_{elt} = alloc(p, q)$ *with* $p < id_{elt} < q$

— *delete($id \in \mathcal{I}$):-* $\mathcal{D}/\{(elt, id)\}$

DEFINITION 2 (VARIABLE-SIZE IDENTIFIER). *A variable-size identifier id is a sequence of numbers $id = [p_1.p_2 \dots p_n]$ which can designate a path in a tree*[1].

In Figure 1, we represent a document as a tree where each identifier is a path from the root to a leaf. In this example, each level has a maximum capacity (arity of the tree node) set to 100. A leaf is an element of the sequence. For instance, [10.13] is an identifier referencing the element b. Assume a user wants to insert an element z between two existing elements identified by p and q:

— if $p = [11]$ and $q = [14]$, there is room for insertion. Both identifiers [12] and [13] are valid choices for the new element.

— if $p = [14]$ and $q = [15]$, there is no room at this level. Since the model does not have further levels, the allocation function *alloc* initiates a new level. Then, it chooses among this bunch of newly available identifiers: between [14.0] and [14.99].

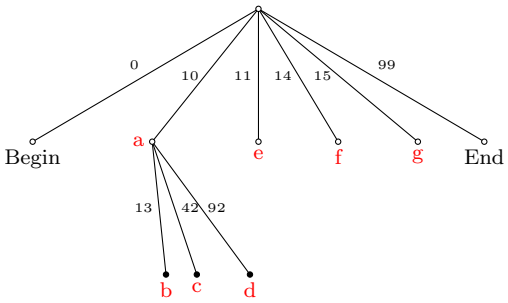
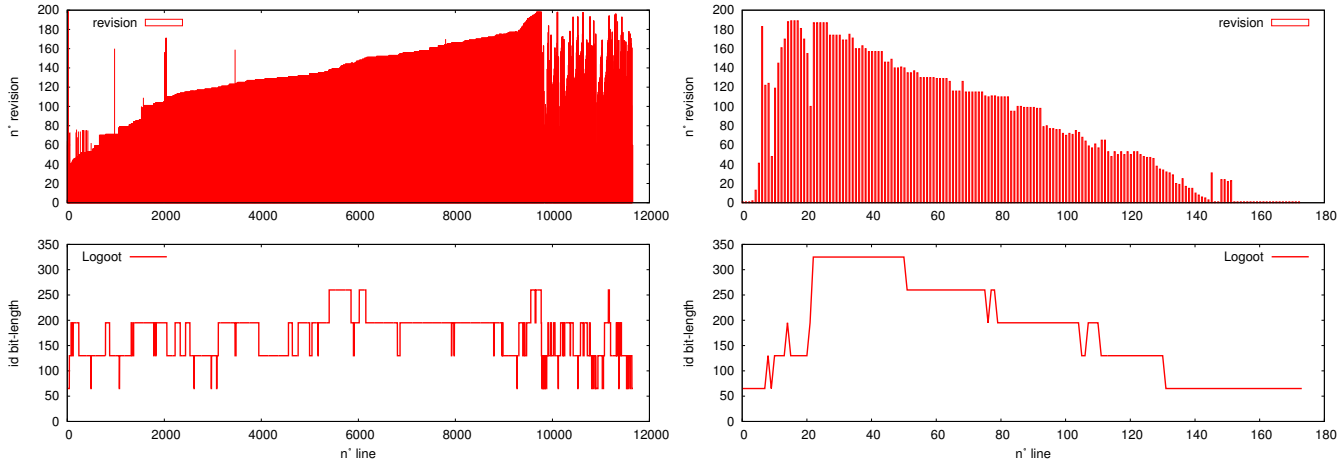


Figure 1: Underlying tree model of a variable-size identifiers sequence CRDT. Depth-1 contains four identifiers [10], [11], [14] and [15] labeling the elements a, e, f and g respectively. Also, depth-1 contains the bounds of the sequence $\langle [0], Begin \rangle$ and $\langle [99], End \rangle$. Depth-2 contains three identifiers [10.13], [10.42], and [10.92] labelling b, c and d respectively.

2.1 Allocation strategies

The Logoot paper [21] already highlighted the importance of allocation strategies (*alloc*). Indeed, experiments concerned two strategies. (1) *Random*: randomly choosing between the identifiers of the two neighbours. It delivers poor performance because the identifiers quickly saturate

1. Identifiers should include site ID to ensure the uniqueness property. However, for clarity purposes and in order to focus on allocation strategy, we did not include any site ID in this definition.



(a) A page of 12k lines mostly edited at the end.

(b) A page of 170 lines mostly edited at the beginning.

Figure 2: Experiments made on a Wikipedia pages. The top figure shows the spectrum of the page (revision number of each line). The bottom figure shows the bit-length of the identifier assigned to each line. The allocation strategy is *boundary* from the Logoot approach.

the space, resulting in the creation of new levels. As consequence, the size of identifiers grows quickly. (2) *Boundary*: randomly choosing between the identifiers of the two neighbours bounded by a *boundary* maximum value. The strategy allocates the new identifiers closer to their preceding identifier. Of course, it works well when the editions are performed right-to-left.

Figures 2a and 2b show the editing behaviour and the bit-length of allocated identifiers on two Wikipedia pages. The top part shows the spectrum associated with the pages. A spectrum gives an overview of the editing behaviour associated with a page. It gives the revision number of each line of a document, i.e., the relative date of their insert operation. As the left spectrum suggests, most of the insert operations situate the new elements at the end of the document, i.e., the last lines of the page are more recent. On the opposite, the second spectrum shows that most of the insert operations situates the newest elements at the beginning of the document. The bottom figures associate the bit-length of the identifier of each line of the document using the *boundary* strategy. In the first figure, we consider a page of $12k$ lines. Identifiers do not exceed 256 bits and they are well spread between levels $[1-4]$. It leads to a satisfying average of 169.7 bits/id. On the contrary, the editing behaviour on the second document (see figure 2b) that has only 170 lines does not fulfills the right-to-left editing behaviour assumed by the *boundary* strategy. In this case, we observe, on an existing document, the worst-case of linear growth of the size of identifiers. The average bit-length is 172.25 bits/id over 5 levels.

2.2 Issues and motivations

Most of existing CRDTs' allocation strategies make the assumption of right-to-left and top-to-bottom editing behaviour. This strong hypothesis allows better space management but other behaviours may lead to a quick decrease in performance. Therefore, it makes the distributed collaborative editor unsafe.

In order to build an efficient distributed collaborative editor based on a sequence CRDT, we need an adaptive allocation function *alloc*, i.e., an allocation strategy independent of an editing behaviour. *The unpredictability of the editing behaviour makes the allocation of identifiers challenging. At any time, the CRDT knows what happened in the past and the current operations. Still, inferring the upcoming operations is complex if not impossible.*

Definition 3 (Problem statement).
Let $\mathcal{D}$ be a document on which n insert operations have been performed. Let $\mathcal{I}(\mathcal{D}) = \{id | (\_, id) \in \mathcal{D}\}$. The function $alloc(id_p, id_q)$ should provide identifiers such as:

$$\sum_{id \in \mathcal{I}} \frac{log_2(id)}{n} < O(n)$$

The problem statement concerns the allocation function *alloc* which should have a sub-linear upper-bound in its space complexity. Such function would greatly improve the current state of art since the document does not require any additional costly protocol: the average size of identifiers being under an acceptable bound.

3. LSEQ ALLOCATION FUNCTION

LSEQ applies a very simple strategy: each time it creates a new level in the tree between two identifiers p and q, it doubles the base of this depth and it randomly chooses a strategy among *boundary+* and *boundary–*. *boundary+* allocates from p plus a fixed boundary, *boundary–* allocates from q minus a fixed boundary. The boundary never changes whatever the depth of the tree.

The following idea is the foundation of this approach: as it is complex to predict the editing behaviour, the principle is to sacrifice some depths of the tree with the certainty that the reward will compensate the loss. In other words, if LSEQ chooses the wrong strategy at a given depth, it will eventually choose the right one in the next depths. Since it doubles the base at each new depth, when the right strategy is found, it will overwhelm the cost of the lost depths.

3.1 Base Doubling

Logoot's [21] underlying allocation strategy always uses the same base to allocate its identifiers. With regard to the tree representation, it means that the arity is set to *base*. A high base value is not profitable if the number of insert operations in this part of the sequence is low. On the contrary, keeping a constant base value when the number of insert operations starts to be very high does not allow to fully benefit of the *boundary* strategy. For instance, Figure 2a presents experimental results from a Wikipedia page that has 12k lines which justifies the usage of a large base unlike Figure 2b with only 170 lines. Knowing the dilemma, the objective is to adapt the base according to the number of insertions in order to make a better reflection of the actual size of the document. Since it is impossible to know *a priori* the size of the document, the idea is to start with a small base due to the empty sequence, and then to double it when and where necessary, i.e. when the depth of identifiers increases.

Doubling the base at each depth implies an exponential growth of the number of available identifiers. Thus, the model corresponds to the exponential trees [3, 17, 2] and consequently it benefits of their complexities. An exponential tree of depth k can store up to $N_k = N_{k-1} + k * k!$ identifiers where $N_1 = base$. In other words, the arity of a node depends of its depth: a node has twice more children than its parent node, and the root has *base* children.

Knowing this exponential tree model, the binary representation of the identifier is $\Sigma_{i=1}^{id.size} b * 2^i$ where b is the initial base (conveniently a power of 2). Practically, if the initial base is 2^4 then, there are 2^{4+1} possibilities to choose an identifier at depth 1, 2^{4+2} at depth 2, etc.

The base doubling relies on the following assumption: the lack of space triggers the growth of identifiers. Therefore, an inefficient allocation strategy will entail an excessive growth of the identifier size as the system doubles the base frequently and the additional depths are more and more costly.

3.2 Allocation Strategies

[21] introduced two allocation strategies: *boundary* and *random*. In the experiments, the former outperforms the latter. However, the *boundary* strategy is heavily application dependent. If a user mainly performs insert operations at the end of the document the allocation will perform well. However, front editing will cause a quick linear growth of the size of identifiers.

With LSEQ, we introduce the allocation strategy named *boundary–*. Basically, this strategy is the opposite of the original *boundary* strategy. In this paper, we rename *boundary* to *boundary+*. Let us consider an insert operation between two elements with the identifiers p and q. While the *boundary+* strategy preferably allocates a position near the preceding identifier p, the *boundary–* strategy allocates a position near the succeeding identifier q. Indeed, *boundary–* starts from position q and subtracts a boundary value instead of starting from position p and adding a boundary value. The arithmetic operation explains the names given to these strategies. Figure 3 shows the results obtained by these two strategies with the same neighbours and random value. The left figure shows the *boundary+* strategy which ends up with [50.11] while the right figure shows the *boundary–* strategy which ends up with [50.89]. They leave free space for future insertions of 88 identifiers at the end and at the beginning respectively.

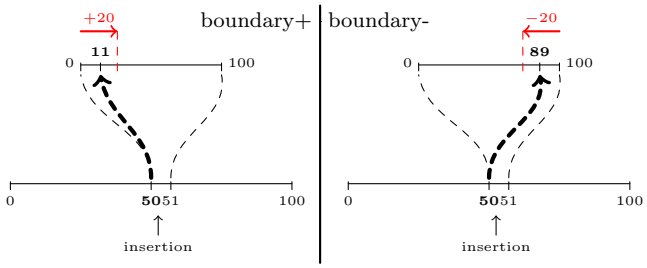


Figure 3: Choice of the digit part of identifiers in *boundary+* (left) and *boundary-* (right). In both cases: constant base is set to 100, boundary value is set to 20 and the random number is 11. The results are [50.11] (*boundary+*) and [50.89] (*boundary-*).

As expected, while the *boundary+* algorithm handles the end editing, the *boundary–* algorithm aims the front editing. They both have an antagonist weakness. Thus, *boundary–* cannot be used alone safely, just like *boundary+*.

3.3 Strategy choice

Current variable-size sequence CRDTs rely on a unique strategy that is not versatile in the sense that it does not adapt to all editing behaviour. As it is impossible to know *a priori* the editing behaviour and then, obtain the best strategy for every sequence, LSEQ randomly alternates between *boundary+* and *boundary–*. Thus, when LSEQ increases the identifier size, it has $\frac{1}{2}$ chance to choose either *boundary+* or *boundary–*. This kind of choice implies lost depths but the main idea is: some depths are lost indeed, nevertheless it is acceptable if the reward compensates the losses.

Algorithm 1 details the allocation function LSEQ. The departure base is set to 2^4 (depth-0) and the *boundary* to 10. The collection $\mathcal{S}$ stores the strategy choices. It starts empty. Three parts compose the algorithm. (1) The first part processes the interval between the two identifiers p and q at each depth until one identifier at least can be inserted. The step limits the interval where *alloc* will allocate the new identifier. (2) The second part determines the allocation strategy. If the function did not allocate any identifiers at this depth yet, it randomly chooses among *boundary+* and *boundary–*. Then it saves this choice for future decisions in $\mathcal{S}$. (3) The final part of the algorithm constructs the new identifier. Depending on the strategy, it draws a random value using the *step* previously processed, and adds/subtracts this value to the $prefix$ of p/q at the wanted depth. The $prefix$ function takes an identifier id as argument, and copies it until it reaches *depth*. If the identifier size is smaller than the requested depth, the function appends a zero to the copy for each missing depth. Each number in the sequence that composes the identifier must be carefully encoded in the base depending on the depth. Line 35 refers to $base(cpt)$. It is a very simple function that computes the base value at a given depth (cpt). Thus, $0_{base(cpt)}$ means that the binary representation of 0 uses $log_2(base(cpt))$ bits. Consequently, the add and the subtract operations do not require additional computation compared to regular arithmetic operations.

Figure 4 illustrates the allocation strategy LSEQ by showing its underlying tree model. First the empty sequence

Algorithm 1 LSEQ allocation function

```
1: let boundary := 10;        ▷ Any constant
2: let S := {};        ▷ map<depth,boolean>
3:       ▷ true: boundary+
4:       ▷ false: boundary−
5:
6: function ALLOC(p, q ∈ I)
7:     let depth := 0;
8:     let interval := 0;
9:     while (interval < 1) do        ▷ Not enough for 1 insert
10:        depth + +;
11:        interval := prefix(q, depth) − prefix(p, depth) − 1;
12:    end while
13:    let step := min(boundary, interval);        ▷ Process the
   maximum step to stay between p and q
14:
15:    if not(S.exist(depth)) then        ▷ add the new entry
16:        let rand := RandBool();
17:        S.set(depth, rand);
18:    end if
19:    if S.get(depth) then        ▷ boundary+
20:        let addVal := RandInt(0, step) + 1;
21:        let id := prefix(p, depth) + addVal;
22:    else        ▷ boundary−
23:        let subVal := RandInt(0, step) + 1;
24:        let id := prefix(q, depth) − subVal;
25:    end if
26:    return id;
27: end function
28:
29: function PREFIX(id ∈ I, depth ∈ N*)
30:    let idCopy := [ ];
31:    for (cpt := 1 to depth) do
32:        if (cpt < id.size) then        ▷ Copy the value
33:            idCopy = idCopy.append(id.at(cpt));
34:        else        ▷ Add 0 encoded in the right base
35:            idCopy = idCopy.append(0_base(cpt));
36:        end if
37:    end for
38:    return idCopy;
39: end function
```

contains only two identifiers: the beginning ([0]) and the end ([31]). The sequence needs three additional identifiers between [0] and [31]. First, LSEQ randomly assigns *boundary+* as allocation strategy to the depth-1. Then, it employs this strategy to allocate the three new identifiers ([9], [10], [23]). The randomness makes the first and second elements very close in terms of identifier distance. Unfortunately, the sequence requests three other identifiers between these two. Consequently, the depth has to grow to contain these new elements. Since LSEQ have not used any strategy at this depth yet, it must randomly choose one. Here, the choice is *boundary–*. Therefore, this strategy allocates the three new identifiers. Furthermore, the underlying exponential tree model extends the number of possible identifiers to 64. In this example, the resulting fresh identifiers are [9.32], [9.51] and [9.60].

This example highlights the principle of LSEQ. Figure 4 depicts an exponential tree model that clearly grows in arity over depths. It means more and more available identifiers when the tree grows. This design aims to adjust the depth of the tree to the number of insert operations. The next section aims to demonstrate experimentally that LSEQ achieves sub-linear space complexity in extreme setups and also outperforms state-of-the-art CRDTs on real documents.

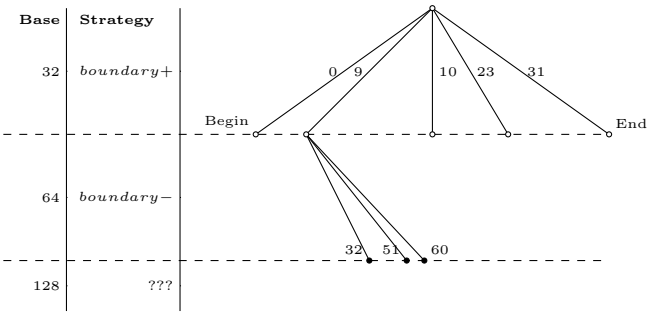


Figure 4: Underlying tree model of LSEQ containing three identifiers at depth-1. The randomness makes the first and second elements very close regarding their identifiers ([9] and [10]). The sequence requests three other elements between these two. The chosen strategy is *boundary–* and since LSEQ doubles the base at each depth, it allocates the fresh identifiers closer of [10.64].

4. EXPERIMENTATION

This experimentation section is comprised of two parts. The first part focuses on highlighting the behaviour of LSEQ on extreme cases. The measurements capture the effect of a large number of insert operations on the identifier sizes. We synthesized different editing behaviours. Analyses are made step by step to bring out the contribution of each component to LSEQ. Previous experiments [1, 12, 21] focused on average setups and did not consider such extreme setups.

The second part of experiments aims to validate if LSEQ also performs well on average setups. In order to do so, we compare Logoot identifiers to LSEQ identifiers on representative Wikipedia pages with antagonist editing behaviours. We choose Logoot as it delivers overall best performances for variable-size sequence CRDTs according to [1].

The experiments focus on the digit part of identifiers. Indeed, the source and clock part of identifiers are common to all the variable-size identifiers approaches. They do not impact on the complexity and can be drastically compressed. Consequently, it is the digit part that reflects the significant improvements.

In order to evaluate LSEQ performance, we developed a Java framework called LSEQ and released the source on GitHub platform under the terms of the GPL licence [2].

4.1 Synthetic Documents Experiments

We designed three experimental setups of synthetic sequences, namely monotone editing behaviour in one position (first and last position), and totally random insertions. The monotonic insertions algorithms choose a particular element and continuously insert new elements before/after this element. For the front editing, it targets the beginning of the document and inserts continuously after it. The end editing, it targets the end of the document and inserts continuously before it. The random behaviour randomly inserts elements in the range $[0 - doc.size[$. The insertions algorithms perform a large number of insert operations on the sequence (up to 10^6). Furthermore, each operation only concerns one element at a time.

In these experiments, we measure the average bit-length of the digit part of identifiers on four different configurations.

2. `https://github.com/Chat-Wane/LSEQ`

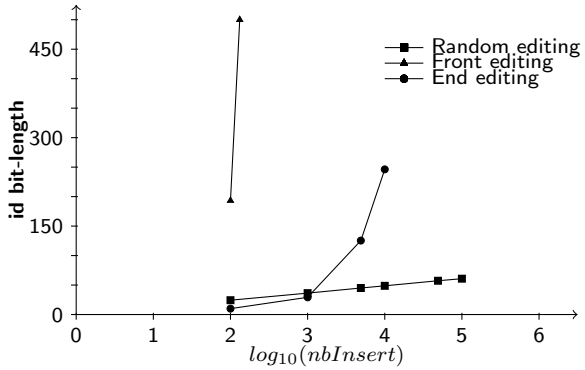


Figure 5: Simple *boundary+* setup (**B**) with $base = 2^{10}$ and $boundary = 10$

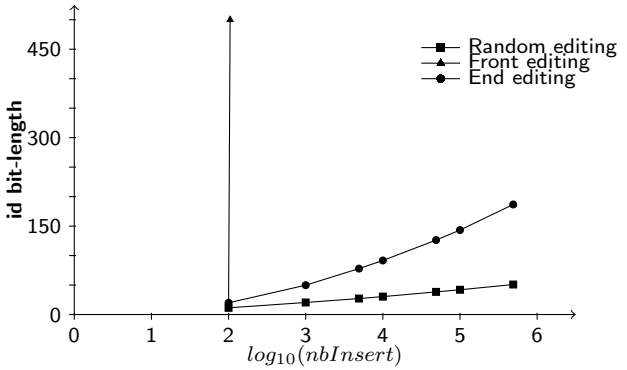


Figure 6: Base doubling setup (**D**) ($base = 2^{4+id.size}$, $boundary = 10$)

In order of appearance: a simple *boundary+* strategy (**B**), a base doubling (**D**) at each new depth, a Round-Robin strategy choice (**RR**) and a random strategy choice with base doubling (**LSEQ**).

Boundary **B** *Experiment.*

OBJECTIVE: to show that *boundary+* does not adapt itself neither to any monotonic editing behaviour nor to the number of insert operations. The expected space complexity is linear compared to the number of inserts in any monotonic editing behaviour. The random editing should lead to a logarithmic size of identifiers.

DESCRIPTION: the measurements concern the average bit-length of the digit part of identifiers. The checkpoints are 100, 1000, 5000, 10000, 50000, 100000 insert operations. The experimental setup is **B** with the following parameter values: a *boundary+* strategy with $boundary = 10$ and a constant $base = 2^{10}$. It corresponds to the Logoot approach with lower values.

RESULTS: Figure 5 shows on the x-axis the number of insertions with a logarithmic scale and on y-axis the average bit-length of identifiers. As expected the identifiers size grows when the number of insertions increases. **B** handles the random editing behaviour with a logarithmic average growth of its identifiers. However, with both front and end editing behaviour, the curve is linear compared to the number of insertions. The end editing remains acceptable in comparison of front editing, but the linear growth would eventually lead to the need of a costly re-balance protocol.

REASONS: The front and end editing behaviours tend to unbalance the underlying tree model of **B**. The *boundary+* allocation strategy has been designed to handle edition at the end. It reserves more space for identifiers at the end, predicting future insertions. The obverse is less space for identifiers at front, therefore the front editing behaviour unbalances the tree even more quickly (leading to a worst-case space complexity of the total identifier size of $O(nb\_insert^2)$). For the same reason, the random editing behaviour leads to logarithmic space complexity: the tree model is balanced.

Base doubling **D** *experiment.*

OBJECTIVE: to show that **D** is not suitable in any case because it does not adapt on the editing behaviour. However, it constitutes an improvement over **B** due to its scalability in terms of insertions number. Indeed, it has a sub-linear upper-bound when the editing behaviour is the expected one. Since **D** uses a *boundary+* allocation strategy, the sub-linear upper-bound is on the end editing. On the other hand, the expectation on the front editing is even worse than the first experiment (with **B**). The random editing should stay with its logarithmic shape unchanged.

DESCRIPTION: like the previous experiment, this experimentation concerns the average bit-length of identifiers. The **D** setup provides the new identifiers. *boundary+* and base doubling compose this setup. The variables are $boundary = 10$ and a base starting from $base = 2^{4+depth}$. The measures are taken at 100, 1000, 5000, 10000, 50000, 100000, 500000 insertions.

RESULTS: Figure 6 shows on the x-axis the number of insertions on a logarithmic scale, and on the y-axis the average id bit-length. Like **B**, **D** provides constantly growing identifiers. When the editing behaviour is the expected one, the growth is sub-linear compared to the number of insertions. Otherwise, the growth is quadratic. Given this, **D** alone is better than **B** when the current editing behaviour is known. In our context where we have no prior knowledge of the editing behaviour, **D** alone is unsafe.

REASONS: the base doubling assumes that a high number of insertions triggered the creation of previous levels in the tree. Thus, it enlarges the number of available identifiers in the new level. If the insertions saturated the previous levels, then it verifies this hypothesis, resulting in an efficient allocation. Of course, if the base doubling hypothesis is false, in the worst case, each new level will contain only one identifier. Each one of these identifiers will have a space complexity equal to $\sum_{i=1}^{n} (log_2(b) + i)$ where n is the number of insertions and b is the departure base.

Round-Robin alternation **RR** *experiment.*

OBJECTIVE: to show that a Round-Robin alternation between *boundary+* and *boundary–* provides identifiers with a linear upper-bound and consequently does not scale as regards the number of insertions. However, it is an improvement over **B** and **D**: with no *a priori* knowledge **RR** avoids the trivial worst case.

DESCRIPTION: the experiment focuses on the average bit-length of the digit part of identifiers. The configuration is

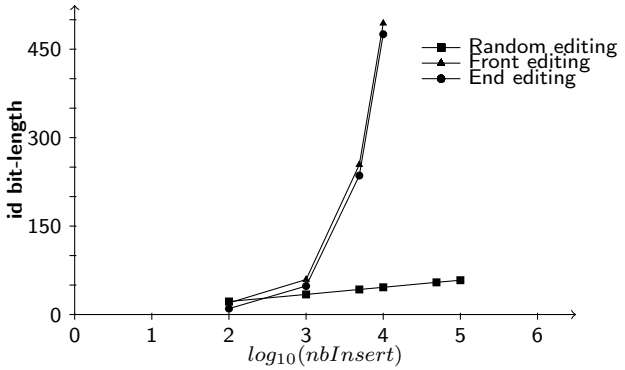


Figure 7: Round-Robin (**RR**) alternation of strategies *boundary+* and *boundary–* ($base = 2^{10}$; $boundary = 10$)

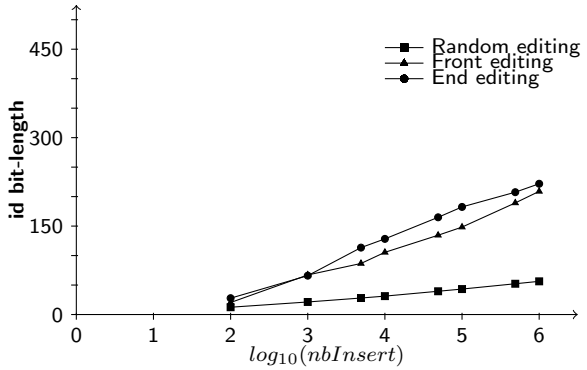


Figure 8: Random alternation (**LSEQ**) of strategies *boundary+* and *boundary–* ($base = 2^{4+id.size}$; $boundary = 10$)

a **RR** setup with two allocation strategies *boundary+* and *boundary–*. The parameters value are $boundary = 10$, and a constant $base = 2^{10}$. The measures are taken at 100, 1000, 5000, 10000, 50000, 100000 insertions.

Results: Figure 7 shows on the x-axis on a logarithmic scale the number of insertions performed on the sequence. The y-axis presents the average bit-length of the digit part of identifiers. While on the random editing behaviour, the identifiers size curve stays in a logarithmic shape, front and end editing are both in linear shape. These observations mean that like **B**, **RR** does not adapt to the number of insertions, and, on the opposite of **B** and **D** it avoids the trivial worst case of front edition. Since every collaborative editing behaviour is a composition of front, end and/or random edition, **RR** is more predictable. However, **RR** remains unsafe because it does not take into account the large number of monotonic insertions.

Reasons: compared to **B**, the average bit-length of identifiers grows two times faster in the case of the end editing behaviour. Indeed, the **RR** alternation of strategies avoids the trivial worst case with the inappropriate editing behaviour (in front). This improvement comes at a cost: half the time **RR** does not employ the well suited strategy, justifying the multiplicative factor of two. The linear space complexity of **RR** stays unchanged compared to **B**. Consequently, **RR** cannot adapt to high number of insertions. **RR** does not overwhelm the loss of one level by the gain obtained in succeeding levels.

LSEQ *experiment.*

Objective: to show that LSEQ remedies both problems of (i) editing behaviour dependence and (ii) the non-adaptive behaviour as regards the number of insert operations. The expected space complexity of the identifiers is sub-linear compared to the number of insertions, both in front and end editing. The random editing stays with the logarithmic behaviour.

Description: we measure the bit-length of the digit part of identifiers. The LSEQ approach provides the identifiers. It lazily and randomly assigns either *boundary+* or *boundary–* to each depth. The *boundary* parameter is set to 10 and the base is doubled over depths. Its departure value is $base = 2^4$. The checkpoints of measurement are 100, 1000, 5000, 10000, 50000, 100000, 500000, 1000000 insertions.

Results: Figure 8 shows the average bit-length of **LSEQ** identifiers on the y-axis. The x-axis represents the number of insertions on a logarithmic scale. Both front and end editing are now sub-linear compared to the number of inserts. On this setup, the curves are poly-logarithmic. The average values are close of the **D** setup. It means that LSEQ loses some depths, but future insertions quickly amortize them. More precisely, it means that the base doubling is profitable enough to compensate the previous lost depths. These two changes make LSEQ a suitable safe allocation strategy for sequences.

Reasons: base doubling **B** performs well if its hypothesis is true, i.e., a high number of insertion triggered the creation of levels. The random choices of strategy among *boundary+* and *boundary–* makes the base doubling hypothesis true with a probability of $\frac{1}{2}$. So eventually, **LSEQ** will obtain the expected gain of base doubling. This gain is high enough to overwhelm the loss of previous levels. It results in a sub-linear upper-bound on the space complexity of **LSEQ**.

4.2 Real Documents Experiments

In previous section, we demonstrate experimentally a sub-linear upper-bound for LSEQ. Next, we aim to confirm the LSEQ properties on real documents. As Logoot delivers best overall performances according to [1], we compare LSEQ with Logoot on Wikipedia documents as previously done in [22].

We select Wikipedia documents with a large amount of lines, with front editing and end editing spectrum. We compare the following two setups: (1) Logoot (**L**) as [21] originally described it, (2) a composition of base doubling and Round-Robin strategy choice (i.e. equivalent to LSEQ) (**LSEQ**$^{\approx}$).

End Editing in Wikipedia.

Objective: to confirm that **LSEQ**$^{\approx}$ (and consequently LSEQ) brings an improvement on the allocation of identifiers, even in cases where previous approaches are known to be good.

Description: the Wikipedia page chosen [3] to run experiments contains a high amount of lines, mainly added at

3. `http://fr.wikipedia.org/wiki/Liste_des_bureaux_de_poste_français_classés_par_oblitération_Petits_Chiffres`

the end. The nature of stored data explains the editing behaviour: a list of postal marking ids applied to letters. Experiments concern two configurations. (1) **L** with a single *boundary+* strategy, and parameters set to $base = 2^{64}$ and $boundary = 1M$, (2) **LSEQ$^{\approx}$** that alternates the two allocation strategies *boundary+* and *boundary–*, and parameters set to $base = 2^{4+depth}$, $boundary = 10$.

RESULTS: Figure 9a shows that, on this document, the bit-length of **LSEQ$^{\approx}$** identifiers is lower than the ones of **L** in the whole document. Table 1 reflects these results: the average bit-length of **LSEQ$^{\approx}$** identifiers is 2.7 times lower than **L** identifiers in spite of the fact that the average size of **LSEQ$^{\approx}$** identifiers (i.e. number of depths) is 2.36 times higher. Therefore, **LSEQ$^{\approx}$** seems to be better suited than Logoot on documents with end editing. It corroborates the observations made in section 4.1.

REASONS: when **L** has to increase the depth of its identifiers, it allocates a large additional space. Each new depth costs 64 bits. It supposedly handles $2^6 4$ more elements. However, the adding of depth happens very quickly when the editing behaviour is not exactly as expected. In particular, the spectrum of the document shows very erratic insertions at the end (in the references and external links part). On the other hand, **LSEQ$^{\approx}$** tries to allocate "when it is needed". It explains why minor editing behaviour changes do not affect a lot the identifiers size. Furthermore, the base doubling of **LSEQ$^{\approx}$** adapts progressively the allocations to the high number of insertions.

		L	**LSEQ$^{\approx}$**
id-length	avg	2.65	6.25
	max	4	12
id-bit-length	avg	169.7	61.24
	max	256	150

Table 1: Numerical values of experiments on the Wikipedia page edited at the end (corresponding to Figure 9a).

Front Editing in Wikipedia.

OBJECTIVE: to highlight the importance of alternating the allocation strategies in **LSEQ$^{\approx}$**. In other words, the *boundary+* strategy of **L** is not sufficient to provide a safe allocation system. Finally, to show that **LSEQ$^{\approx}$** outperforms **L** on documents edited at the beginning.

DESCRIPTION: we choose the Wikipedia page [4]. Since it is a "talk" page, it provides a discussion space. The users mostly inserted elements at the beginning of the document. Once again, we make the measurements on two configurations. (1) **L** with a single *boundary+* strategy, and parameters set to $base = 2^{64}$ and $boundary = 1M$, (2) **LSEQ$^{\approx}$** with the two allocation strategies *boundary+* and *boundary–*, and $base = 2^{4+depth}$, $boundary = 10$.

RESULT: unsurprisingly, the figure 9b shows that using **L**, the identifiers bit-length increases very fast in the beginning of the document while it quickly stabilizes when **LSEQ$^{\approx}$** is used. In Table 2, we observe that the average identifiers bit-length of **LSEQ$^{\approx}$** is 3.31 times lower than the one of **L**. The alternation of strategies allows **LSEQ$^{\approx}$** to quickly find a depth where allocation of identifiers will be efficient, and thereby to amortize previous depths where some spaces could have been wasted. These observations confirm the results of section 4.1.

REASONS: **LSEQ$^{\approx}$** does not favour any editing behaviour thanks to its allocation strategies. On the opposite, **L** uses an allocation strategy designed to support end editing, thus, when the antagonist behaviour arises, the identifiers size grow very fast.

		L	**LSEQ$^{\approx}$**
id-length	avg	2.69	5.29
	max	5	8
id-bit-length	avg	172.25	51.99
	max	320	84

Table 2: Numerical values of experiments on the Wikipedia page edited at the beginning (corresponding to Figure 9b).

4.3 Synthesis

Experiments evaluated the contribution of each part of LSEQ allocation function. They demonstrated that each isolated component cannot achieve sub-linear space complexity. However, their composition with random choice among *boundary+* and *boundary–* and a base doubling can achieve sub-linear space complexity in extreme setups. We also observe this gain on real documents. Consequently, LSEQ is suitable for building distributed collaborative editors that deliver better performance and in a larger scope of usage than state of art.

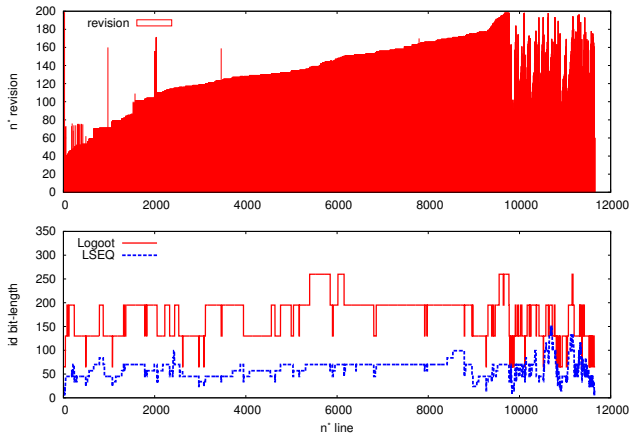
5. RELATED WORK

Popular distributed collaborative editors such as Google Docs [10] rely on Operational Transformation approach (OT) [18, 19]. OT-based and CRDT-based distributed editors follow the same global scheme of optimistic replication, i.e., generate operations without locking, broadcast to others replicates and re-execute. OT and CRDT mainly differ in their complexities: (i) OT-based editors have constant-time complexity at generation time and a complexity of $O(|H^2|)$ at re-execution time where H is the log of operations. Performance of OT closely depends on the number of concurrent operations present in the system. (ii) LSEQ sequence CRDT has a complexity of $O(k)$ for generation time and $O(k * log(n))$ for re-execution time where n is the number of elements presents in the document and k is proportional to size of identifiers. Unlike OT, the state of the document mainly determines the CRDT performance. LSEQ significantly improves the performance of the sequence CRDTs by keeping k small.

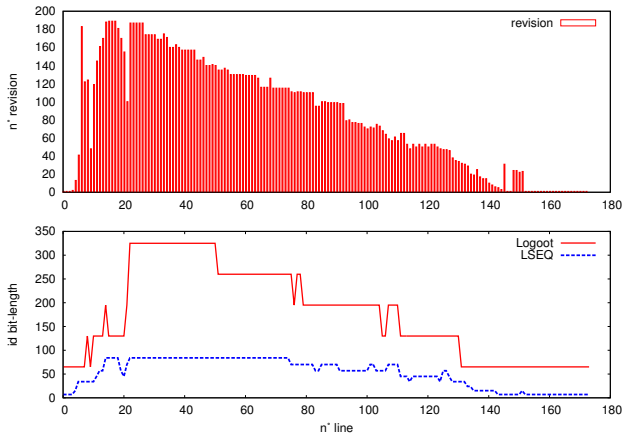
The tombstone class of sequence CRDT includes WOOT [11], WOOTO [20], WOOTH [1], Treedoc [12], CT [7], RGA [13], [23]. In these approach, tombstones (or "death certificates") mark the deleted elements. They provide a simple solution to solve problems of concurrent delete. A clear advantage is to only require fixed-length identifiers, nonetheless the space complexity of tombstone-based sequence CRDTs is linear compared to the number of insert operations performed on the document.

Safely garbaging tombstones in a distributed system is costly because it requires obtaining a consensus for this decision among all participants. In [13, 9], they proposed some solutions related to the garbage collecting mechanism in order to rebalance and/or purge the model of the CRDT. The

4. `http://en.wikipedia.org/wiki/Template_talk:Did_you_know`



(a) The Wikipedia page has 12k lines and is mostly edited at the end. The average bit-lengths of identifiers are 168.7 and 61.24 for **L** and **LSEQ**$^{\approx}$ respectively.



(b) The Wikipedia page has 170 lines and is mostly edited at the beginning. The average bit-lengths of identifiers are 172.25 and 51.99 *bit/id* for **L** and **LSEQ**$^{\approx}$ respectively.

Figure 9: The top spectrums reflect the editing behaviour performed on Wikipedia pages. The bottom figures shows the identifier bit-length assigned to each line. Two configurations: Logoot (**L**) and Round-Robin with base doubling (**LSEQ**$^{\approx}$).

purge [13] of tombstones requires a full vector clock to keep track of updates on other replicas and to be able to safely remove the tombstones. The *core nebula* [9] approach intends to make the consensus reachable, but constrains the topology of the network and uses an expensive *catch up* algorithm.

The variable-size identifiers class of CRDT includes Logoot [21] and Treedoc [12]. These CRDTs use growing identifiers to encode the total order among elements of the sequence. In the worst case, the size of identifiers is linear in the total number of insert operations done on the document [1]. Logoot and Treedoc [12] have different allocation strategies. Treedoc has two allocation strategies: (i) the first strategy allocates an identifier by directly appending a bit on one of its neighbour identifier. (ii) The second strategy increases the depth of this new identifier by $\lceil log_2(h) \rceil + 1$ (where h is the highest depth of the identifiers already allocated) and allocates the lowest value possible with this growth, in prevision of future insertions.

Logoot's *boundary* strategy and Treedoc's second strategy are very similar, both in their goals and their weaknesses. They assume an editing behaviour in the end, and therefore they become application dependent. Compared to Logoot and Treedoc, LSEQ is adaptive and significantly enlarges the applicability of sequence CRDTs.

In [1], they compared most sequence CRDTs and one OT in an experimental setup. RGA and Logoot obtained best overall performances. In this paper, we completed experiments with more extreme cases and demonstrated that LSEQ outperforms Logoot.

6. CONCLUSION

In this paper, we presented an original allocation strategy for sequence CRDTs called LSEQ. Compared to state of art, LSEQ is adaptive, i.e., it handles unpredictable different editing behaviour and achieves sub-linear space complexity. Consequently LSEQ does not require a costly protocol to garbage or re-balance identifiers, and is suitable for building better distributed collaborative editors based on sequence CRDTs.

Three components compose LSEQ: (1) a base doubling, (2) two allocation strategies *boundary+* and *boundary–*, (3) a random strategy choice.

Although each component cannot achieve sub-linear complexity, the conjunction of three components provides the expected behaviour. Experiments show that even if LSEQ makes a bad strategy choice for one level in the tree, this choice will be overwhelmed by the gain obtained at next levels.

The LSEQ approach is generic enough to be included in other variable-size sequence CRDTs. Current experiments were done with a Logoot basis because it does not require tombstones and therefore is less dependent of the editing behaviour. But we believe that Treedoc's heuristic could be improved with this allocation strategy.

Future works include a formal demonstration of the empiric poly-logarithmic upper-bound in space complexity of LSEQ which implies a probabilistic study of its worst-case. The idea is to prove that its probability of happening is negligible. We also plan to study if concurrency affects LSEQ results, i.e., if each site makes different allocation choices concurrently, does it impact LSEQ performances? Finally, we aim to study if using documents spectrum knowledge and machine-learning approaches can outperform random strategy choice.

7. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their comments and suggestions which not only strenghten this work but also led to answer some thorny issues about future work.

References

[1] M. Ahmed-Nacer, C.-L. Ignat, G. Oster, H.-G. Roh, and P. Urso. Evaluating CRDTs for Real-time Document Editing. In ACM, editor, *11th ACM Symposium on Document Engineering*, pages 103–112, Mountain View, California, États-Unis, Sept. 2011.

[2] A. Andersson. Faster deterministic sorting and searching in linear space. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 135–141. IEEE, 1996.

[3] A. Andersson and M. Thorup. Dynamic ordered sets with exponential search trees. *J. ACM*, 54(3), June 2007.

[4] 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, pages 399–407, New York, NY, USA, 1989. ACM.

[5] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[6] S. Greenberg and D. Marwood. Real time groupware as a distributed system: concurrency control and its effect on the interface. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 207–217, 1994.

[7] V. Grishchenko. Deep hypertext with embedded revision control implemented in regular expressions. In *Proceedings of the 6th International Symposium on Wikis and Open Collaboration*, WikiSym '10, pages 3:1–3:10, New York, NY, USA, 2010. ACM.

[8] P. R. Johnson and R. H. Thomas. Maintenance of duplicate databases. RFC 677, Jan. 1975.

[9] M. Letia, N. Preguiça, and M. Shapiro. Crdts: Consistency without concurrency control. *Arxiv preprint arXiv:0907.0929*, 2009.

[10] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 111–120. ACM, 1995.

[11] G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for p2p collaborative editing. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 259–268. ACM, 2006.

[12] N. Preguiça, J. M. Marquès, M. Shapiro, and M. Letia. A commutative replicated data type for cooperative editing. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 395–403. Ieee, 2009.

[13] H.-G. Roh, M. Jeon, J.-S. Kim, and J. Lee. Replicated abstract data types: Building blocks for collaborative applications. *Journal of Parallel and Distributed Computing*, 71(3):354–368, 2011.

[14] Y. Saito and M. Shapiro. Replication: Optimistic Approaches. technical report, 2002.

[15] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, Mar. 2005.

[16] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, 2011.

[17] A. Singh and D. Garg. Implementation and performance analysis of exponential tree sorting. *International Journal of Computer Applications ISBN*, pages 978–93, 2011.

[18] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, pages 59–68, New York, NY, USA, 1998. ACM.

[19] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1):63–108, 1998.

[20] S. Weiss, P. Urso, and P. Molli. Wooki: A p2p wiki-based collaborative writing tool. In B. Benatallah, F. Casati, D. Georgakopoulos, C. Bartolini, W. Sadiq, and C. Godart, editors, *Web Information Systems Engineering – WISE 2007*, volume 4831 of *Lecture Notes in Computer Science*, pages 503–512. Springer Berlin Heidelberg, 2007.

[21] S. Weiss, P. Urso, and P. Molli. Logoot: a scalable optimistic replication algorithm for collaborative editing on p2p networks. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 404–412. IEEE, 2009.

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

[23] W. Yu. A string-wise crdt for group editing. In *Proceedings of the 17th ACM international conference on Supporting group work*, GROUP '12, pages 141–144, New York, NY, USA, 2012. ACM.