Logoot-Undo: Distributed Collaborative Editing System on P2P networks

Stéphane Weiss, Pascal Urso, Pascal Molli

Abstract—Peer-to-peer systems provide scalable content distribution for cheap and resist to censorship attempts. However, P2P networks mainly distribute immutable content and provide poor support for highly dynamic content such as produced by collaborative systems. A new class of algorithms called CRDT (Commutative Replicated Data Type), which ensures consistency of highly dynamic content on P2P networks, is emerging. However, if existing CRDT algorithms support the "edit anywhere, anytime" feature, they do not support the "undo anywhere, anytime" feature. In this paper, we present the Logoot-Undo CRDT algorithm, which integrates the "undo anywhere, anytime" feature. We compare the performance of the proposed algorithm with related algorithms and measure the impact of the undo feature on the global performance of the algorithm. We prove that the cost of the undo feature remains low on a corpus of data extracted from Wikipedia.

Index Terms—Collaborative Editing, P2P, Group Undo, Scalability, Optimistic Replication, CRDT.

1 Introduction

Peer-to-peer (P2P) systems represent a significant part of all Internet traffic. They provide scalable content distribution for cheap and resist to censorship attempts [1]. However, P2P networks mainly distribute immutable content and provide poor support for highly dynamic content produced by collaborative systems such as wikis, collaborative editors or version control systems. Our objective is to develop models and algorithms that allow deploying efficiently distributed collaborative systems such as Wikipedia on P2P networks.

Managing highly dynamic contents on P2P networks raises the issue of consistency enforcement with P2P constraints such as the unknown number of peers, the high degree of churning and network failures.

A new class of algorithms called CRDT (Commutative Replicated Data Type) [2], [3], [4] is emerging for ensuring consistency of highly dynamic content on P2P networks. Unlike traditional optimistic replication algorithms [5], they do not require to detect the concurrency between operations in order to ensure the CCI (Causality, Convergence and Intention) consistency [6] required for distributed collaborative systems. CRDT algorithms rely on natively commutative operations defined on abstract data types such as lists or ordered trees. Operations are generated on a site, broadcasted to other sites and simply re-executed. As operations are natively commutative, they do not require a merge algorithm or an integration procedure to enforce convergence consistency.

However, if prior works on CRDT algorithms [2], [4], [7], [8] support the "edit anywhere, anytime" feature, they do not support the "undo anywhere, anytime" feature [9]. Collaborative editing without the undo feature is harmful for the following reasons:

- Undo is a user-required feature. Indeed, users can use the undo feature as a powerful way to recover from their proper errors or even from error made by other users,
- In collaborative editors, when two or more users modify the same data, the system proposes a document based on all modifications. This proposal is a best-effort and may not be the result expected by users. The undo feature is useful to recover from unexpected results,
- We consider a P2P Collaborative Editing Systems (CES) as an open system where anyone can join and leave. Since anyone can join, malicious users can also join. As a result, the undo feature can be used to remove the impact of vandalism acts. For instance, in the Wikipedia article "Georges.W Bush", one third of all modifications are "undo" [10].

Supporting the most general kind of undo is hard to achieve [11], [12], [13], [14] in CES. Indeed, the system must deal with concurrent "do", undo, and redo operations. The naive conception of the undo feature simply relies on using the inverse operation. Unfortunately, such mechanism fails to provide directly a correct undo mechanism. For instance, let's consider a collaborative editor for text document where two users undo concurrently the deletion of a line. If undo is realized through inverse operation, each undo generates an insertion of a line, and the resulting document contains twice the same line. Unfortunately, the correct result [15] is a document containing only one line.

In this paper, we present the Logoot-Undo CRDT algorithm that integrates the "undo anywhere, anytime" feature. We compare the performance of the proposed algorithm with related algorithms and measure the impact of the undo feature on the global performance

- *The authors are affiliated to Nancy-Université, LORIA, FRANCE. E-mail: weiss@loria.fr, urso@loria.fr, molli@loria.fr.*

of the algorithm. We prove that the cost of the undo feature remains low on a corpus of data extracted from Wikipedia.

2 P2P Collaborative Editing System

We make the following assumptions about P2P networks and P2P Collaborative Editing Systems (P2P CES).

A P2P network is composed of an unbounded set of peers. Each peer has the same role. We assume each peer possesses a unique site identifier. Peers can enter and leave the network arbitrary fast bringing churn in the network.

We do not make assumptions about the structure of the overlay network. The network can be structured or unstructured. We assume that, for efficiency and fault-tolerance reasons, the content on the network is replicated. This replication can be either total [16] – i.e., each peer owns a replica – or partial [17], [18]. In CES, to achieve high responsiveness and high concurrency, a user can modify a replica at any time [19]. Local modifications are eventually received by all other peers. We make no assumptions about their propagation time. When a peer receives a modification, it replays it on its local replica. Thus, replicas are allowed to diverge in the short time. This kind of replication is known as *optimistic replication* [5] (or lazy replication).

The dissemination mechanism of the local modification through the P2P network is not specified in this article. For instance, it can be achieved with a scalable causal broadcast [20].

According to [6], a collaborative editing system is considered as correct if it respects the CCI criteria:

Causality: All operations ordered by a precedence relation, in the sense of the Lamport's *happened-before* relation [21], are executed in the same order on every replica.

Convergence: The system converges if all replicas are identical when the system is idle : *eventual consistency* [22].

Intention preservation: The expected effect of an operation should be observed on all replicas. A well-accepted definition of operations intentions for textual documents is :

- *delete* A deleted line must not appear in the document unless the deletion is undone.
- *insert* A line inserted on a peer must appear on every peer. And, the order relation between the document lines and a newly inserted line must be preserved on every peer (as long as these lines exist) [23].
- *undo* Undoing a modification makes the system return to the state it would have reached if this modification was never produced [15].

Finally, we want to build collaborative editing systems respecting CCI consistency model and supporting P2P constraints such as:

Scalability: The system must handle the addition of users or objects without suffering a noticeable loss of performance [24].

Churn: Peers can enter and leave the network at any time.

Unknown number of peers: We consider that knowing the exact number of peer at one time is not a realistic hypothesis in dynamic networks such as P2P networks.

Failures: Peers are subject to failure, i.e., any peer can crash at any time without warning other peers.

3 Related Work

Most CES belong to the Operational Transformation [25] (OT) framework. Some approaches [23], [26], [27] do not support the undo feature while other approaches [9], [28] offer an undo mechanism. OT approaches require to detect the concurrency between operations which is costly to achieve in P2P environments. Most OT approaches use state vectors or central timestamper to provide this feature. As a consequence, they do not scale and do not support network churn.

MOT2 [29] is a P2P pair-wise reconciliation algorithm in the OT approach. Contrary to the other OT approaches, MOT2 does not require the use of state vectors or central service. The algorithm requires transformation functions satisfying some properties [30]. To our best knowledge, the only transformation functions usable with MOT2 are the Tombstone Transformation Functions [31] (TTF) whose main idea is to replace deleted lines by tombstones. An undo feature for MOT2 is proposed in [30]. Unfortunately, to detect the concurrency between operations, MOT2 propagate up to all known operations on each reconciliation. Thus, the dissemination mechanism generates high traffic and leads to low convergence.

WOOKI [32] is a P2P wiki system which is based on WOOT [3]. The main idea of WOOT is to treat a collaborative document as a Hasse diagram which represents the order induced by the *insert* operations. Therefore, the WOOT algorithm computes a linear extension of this diagram. WOOKI barely respects the CCI correction criteria. Indeed, the causality is replaced by preconditions. As a result, the happened-before relation can be violated in some cases. The convergence is ensured by the algorithm and by using tombstones. Since tombstones cannot be removed without compromising consistency, performance degrades during time. In [33], the authors propose an undo feature for WOOKI. In this paper, our goal is to propose a similar feature for Logoot without using tombstones.

TreeDoc [7] is a collaborative editing system which uses a binary tree to represent the document. Deleted lines are also kept as tombstones. The authors propose a kind of "2 phase commit" procedure, called "flatten", to remove tombstones. Unfortunately, such a procedure cannot be used in an open-network such as P2P

environments. However, this approach proposes also an interesting general framework called Commutative Replicated Data Type (CRDT) to build distributed collaborative editors ensuring CCI criteria. More recently [2], the authors propose a new version of TreeDoc. The major improvement resides in a unilateral leaf tombstone removal feature. However, the flatten procedure is still required to keep good performances when the number of edits grows. Unfortunately, the undo feature is not supported.

In [8], the authors propose a distributed replication mechanism in the CRDT framework which ensures the CCI criteria but using tombstones and vector clocks. They do not propose an undo mechanism.

DTWiki [34] is a disconnection tolerant wiki based on TierStore, itself based on Delay-Tolerant Network. Causality is achieved in DTWiki using version vectors. In DTWiki, conflict resolution slightly differs from approaches like WOOKI or TreeDoc. Indeed, in case of concurrent modifications of the same wiki page, DTWiki generates one revision per concurrent modifications. Revisions are not automatically merged, even in case of non-conflicting modifications. On the contrary, all approaches considered above, automatically merged concurrent modifications. The undo feature is not supported.

Distriwiki [35] is a P2P wiki system based on JXTA. Unfortunately, the authors do not discuss of the concurrent updates case. Distriwiki does not support the undo feature.

4 Editing a Logoot-Undo document

Logoot-Undo approach belongs to the CRDT framework [2] whose main idea is to provide a genuine commutativity between concurrent operations. When the data type is an ordered list, such as a text document, commutativity between insertions in the list can be obtained with a unique and totally ordered identifier for each line (or each character).

In this section, we present the part of the Logoot-Undo approach that can be used without undo.

4.1 Logoot-Undo model

The main idea is to associate an identifier to each line. This line identifier must be unique, dense and totally ordered.

Definition 1:
- A *position* is a tuple $\langle i, s, c\rangle$ where i is a digit, s a peer identifier and c a clock value.
- A *line identifier* is a non-mutable list of positions. Let $\langle i_n, s_n, c_n\rangle$ be the last position of a line identifier; s_n is the site identifier of the peer that created the line and c_n the value of its clock during the creation of this line.

We assume that each peer has a unique site identifier and a clock. Any base for the digit i of position tuples can be chosen. We call $BASE$, the chosen base. For instance, in our experiments, we choose $BASE = 2^{64}$, and thus i is an standard unsigned integer. According to the above definition, line identifiers are unique in time and space. To obtain a total order between identifiers, we use the following definition.

Definition 2:
- Let $P = p_0.p_1 \dots p_n$ and $Q = q_0.q_1 \dots q_m$ be two identifiers, we get $P \prec Q$ if and only if $\exists j \leq m.\ (\forall i < j.\ p_i = q_i) \wedge (j = n+1 \vee p_j <_{id} q_j)$
- Let $p = \langle i_1, s_1, c_1\rangle$ and $q = \langle i_2, s_2, c_2\rangle$ be two positions, we get $p <_{id} q$ if and only if $i_1 < i_2$, or $i_1 = i_2$ and $s_1 < s_2$, or $i_1 = i_2$ and $s_1 = s_2$ and $c_1 < c_2$.

Finally, a Logoot-Undo model looks like Figure 1 where $\langle 0, NA, NA\rangle$ and $\langle MAX, NA, NA\rangle$ — with $MAX = BASE - 1$ — are special identifiers marking the beginning and the end of the document.

Identifiers table	Document
$\langle 0, NA, NA\rangle$	
$\langle 131, 1, 4\rangle$	"This is a Logoot-Undo document"
$\langle 131, 1, 4\rangle.\langle 2471, 5, 23\rangle$	"A place between 131 and 131"
$\langle 131, 3, 2\rangle$	"This line was the 2nd made on replica 3"
$\langle MAX, NA, NA\rangle$	

Fig. 1. Logoot-Undo model

The identifiers are stored in the identifier table which is a separate storage from the document lines. As a result, the Logoot-Undo approach does not modify the document model which makes its integration into existing CES easier.

Moreover, the mapping cost between the document and the identifiers table is constant. Indeed, to retrieve the identifier of the i^{th} line, we just have to read the $(i+1)^{th}$ element of the identifier table.

We have now defined our data model, the following sections discuss of the modification this data type. Firstly, we consider modifying the document using regular edit actions. Then we focus on undoing existing modifications.

4.2 Modifying a Logoot-Undo document

We assume that a user can modify the document by inserting or deleting lines. Each operation made to the document implies a modification of the associated identifiers table. We assume that a document and the associated identifiers table are modified through atomic changes.

Then, deleting a line in the document requires to remove the identifier associated to this line. Similarly, inserting a line in the document requires to insert an identifier for this line. When a line is created between a line with identifier x and a line with identifier y, a new identifier z is computed. To respect the intention preservation criterion, we need that $x \prec z \prec y$.

Any strategy to construct z can be chosen : randomly between x and y, at the center of x and y, near to x or to y, ... *Moreover, each peer can choose arbitrary any strategy that respects intentions. A peer can also change its strategy at any time without warning other peers.*

[4] propose a Logoot strategy that generates z randomly between x and y. In the rest of the paper, we refer to this strategy as *"Random Strategy"*. Our experiments on the Wikipedia led us to design another strategy that is random within a boundary.

4.3 Line identifier creation

On many collaborative editing systems such as wiki or control version systems, an edit on a document is not a single operation but a set of operations (called a patch). Lines inserted by a patch are often contiguous or group by contiguous blocks.

To apportion the line identifiers of a block of size N, we define the function generateLineId(p,q,N,$boundary$,s) that, on a site s, generates N identifiers between the line identifier p and the line identifier q. The algorithm uses numbers in base-$BASE$. To obtain short line identifiers, the generation algorithm firstly selects the smallest equal length prefixes of p and q spaced out at least N. Then it apportions randomly the constructed identifiers using $boundary$ to limit the distance between two successive line identifiers. Let's call this strategy: *"Boundary Strategy"*. The randomization helps to restrain two different replicas to generate concurrently the same choice, and thus to reduce the growing rate of the lines identifier.

```
function generateLineId(p, q, N, boundary, site)
  let list := ∅,
    index := 0,
    interval := 0;
  while (interval < N)
    index++;
    interval := prefix(q, index) − prefix(p, index) − 1;
  endwhile                    // Finds a place for N identifiers
  let step := min(interval/N, boundary)
    r := prefix(p, index);
  for j:=1 to N do
    list.add(constructId(r + Random(1, step), p, q, site));
    r := r + step;
  done                        // Constructs N identifiers
  return list;
end;
```

The function prefix(p, i) returns a *number* in base-$BASE$. The digits of this number are the i^{th} first position digits of p (filled with 0 if $|p| < i$). The function constructId(r, p, q, $site$) is defined beside.

```
function constructId(r, p, q, site)
  let id := {};
  for i:=1 to |r| do
    d := i^th digit of r
    if (d = p_i.digit) then
      s := p_i.siteid; c := p_i.clock
    elseif (d = q_i.digit) then
      s := q_i.siteid; c := q_i.clock
    else
      s := site.identifier; c := site.clock++
    fi ;
    id := id . ⟨d, s, c⟩;
  done;
  return id;
end;
```

For instance, assume that $BASE = 100$ and $boundary = 10$, and there is a site identified by s with a clock value of h that wants to insert N lines between two lines identified by $p = \langle 2,4,7\rangle\langle 59,9,5\rangle$ and $q = \langle 10,5,3\rangle\langle 20,3,6\rangle\langle 3,3,9\rangle$. The generateLineId algorithm firstly finds a place for the N lines between the shortest possible prefixes of p and q.

1) $prefix(p,1) = 2_{100}$ and $prefix(q,1) = 10_{100}$, thus there are 7 (i.e., $10 - 2 - 1$) identifiers available between prefixes of size 1,
2) $prefix(p,2) = 2.59_{100}$ and $prefix(q,2) = 10.20_{100}$, thus there are 760 (i.e., $1020 - 259 - 1$) identifiers available between prefixes of size 2,
3) $prefix(p,3) = 2.59.0_{100}$ and $prefix(q,3) = 10.20.3_{100}$, thus there are 76102 (i.e., $102003 - 25900 - 1$) identifiers available between prefixes of size 3, etc.

Once selected the shortest possible prefixes according N, the algorithm constructs randomly the N identifiers.

1) if $0 < N \leq 7$, the identifiers are apportioned in the set $\{\langle x,s,h\rangle \mid x \in]2,10[\}$ [1],
2) if $7 < N \leq 760$ the identifiers can be generated in the set of the 760 available identifiers : $\{\langle 2,4,7\rangle\langle x,s,h\rangle \mid x \in]59,100[\}$ $\cup \{\langle y,s,h\rangle\langle x,s,h\rangle \mid y \in]2,10[, x \in [0,100[\}$ $\cup \{\langle 10,5,3\rangle\langle x,s,h\rangle \mid x \in [0,20[\}$.
3) etc.

The distance between the identifiers of two successive lines does not exceed $min(interval/N, boundary)$. Thus, the boundary allows grouping the line identifiers and leaving space for subsequent insertions. For instance, in the above example, with $N = 23$, line identifiers are apportioned in the set of the $N * boundary = 230$ first available identifiers (i.e., between p and $\langle 4,s,h\rangle\langle 89,s,h\rangle$) and not the whole 760 like in the "Random strategy" (see Figure 2).

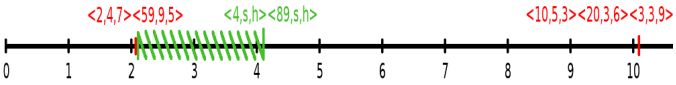

Fig. 2. Logoot identifiers space using boundary

4.4 Propagating "edit" changes

An operation is created for each modification on the document. An operation is either an insertion or a deletion. For each insertion, we create an operation "Insert($id, content$)" where id is the identifier of the inserted line and $content$ the content of the inserted line. Similarly, we create an operation "Delete($id, content$)" for each line deleted from the document.

Operations are grouped in patches and sent to all other replicas to be integrated. We assume that each patch is delivered once and only once to each replica. To achieve this property, we can use a reliable broadcast that use a

1. We use the ISO interval notation: $x \in [a,b[$ means $\{x \mid a \leq x < b\}$

unique identifier to each patch to check for patch duplication. Therefore, for each operation received, a replica has to update both identifiers table and document.

```
function execute(patch)
  for op in patch do
    switch (op):
    case Insert(id, content):
          position := idTable.binarySearch(id);
          document.insert(position, content);
          idTable.insert(position, id);
    case Delete(id, content):
          position := idTable.binarySearch(id);
          if (IdTable[position] = id) then
             document.remove(position, content);
             idTable.remove(position, id);
          fi
    end;
  done
end;
```

Both line insertion and removal can be executed in a logarithmic time according to the number of lines in the document and constant according to the number of users in the P2P network. Indeed, we simply use the binary search algorithm to find the position of the smallest line identifier greater or equal in the identifiers table and insert or delete the content in the document at the same position.

Also, the integration of a delete operation can safely remove the line from the document model, since the total order between remaining lines is not affected. The algorithm only verifies that the line has not been already deleted by a concurrent delete operation.

Logoot supports disconnected work. Indeed, thanks to unique identifiers, operations generated on a replica when the hosting peer is disconnected can be sent when the peer joins back the network. We assume it's also the case when a peer crashes between the local modification and the patch dissemination.

5 Modifying a Logoot-Undo document using "undo" messages

To obtain an undo and redo mechanism that respects the intention preservation criterion, we work at two levels. At the patch level, we add a degree to decide in presence of concurrent undo and redo if the patch must be reversed or reapplied. At the line level, we add a visibility degree to decide in presence of all different operations (insert, delete, and their undo) if the line must be visible or not.

Messages disseminated between Logoot-Undo peers can be 1) a patch, i.e., a set of insert and delete operations, 2) or an undo (or a redo) of a given patch identifier.

5.1 History Buffer

Each site stores patches in a collection HB named *history buffer*. We make no assumptions about the nature of HB[2]. We only assume that HB allows retrieving a patch by its identifier in a reasonable time.

A patch contains at least one operation and contains only operations made by the same site. Each patch is identified by a unique identifier.

5.2 Undoing/Redoing patches

When a user wants to undo a patch, the document must return to a state in which the patch modifications was never performed according to the undo intention definition.

This definition implies two cases:

- the patch is already undone and the document must not be changed,
- the patch must be disabled and its effect must be removed.

Moreover, since the action of undoing a patch is also an edit of the document, users must be able to undo an undo modification, this is called *redo*. Similarly, according to the undo definition, if a patch is already redone, the action of redo has no effect, otherwise, we must re-apply its effect.

Thus, the system must know if a patch is enabled or not. Moreover, the system has to know how many times a patch has been undone or redone as illustrated in Figure 3.

Assume that two sites, called Site1 and Site2, initially on state " ", have received the same patch $P1$ bringing to state "A". Concurrently, both sites decide to undo this patch. Consequently, Site1 generates a message $M1 = undo(P1)$ while Site2 generates $M2 = undo(P1)$. Then, Site2 chooses to redo this patch: $M3 = redo(P1)$. Finally, each site receives each other messages. Site1 has received both "undo" and then the "redo". If the system just knows that $P1$ is undone, the "redo" will reapply $P1$. Unfortunately, this example violates the definition. Indeed, if the message $M2$ was never produced, the only remaining edit is $M1$, then the $P1$ must remain undone.

As a result, we propose to associates an integer called $degree$ to each patch. This $degree$ indicates whether the patch is applied or not in the current state of the document. The degree of a patch is 1 at the reception, it decreases by one at the execution of an undo and increases it by one for a redo. Then, a patch is undone if its degree is strictly inferior to 1. To remove the effect of a patch P, we execute its inverse inverse(P), i.e., we execute delete(i, c) for each insert(i, c), and insert(i, c) for each delete(i, c) in P.

```
function deliver(message)
  switch(message)
  case patch:
    execute(patch);
    HB.add(patch);
    patch.degree := 1;
  case Undo(patchID):
```

2. HB can be for instance a hash table, an ordered list, a causality graph, etc.

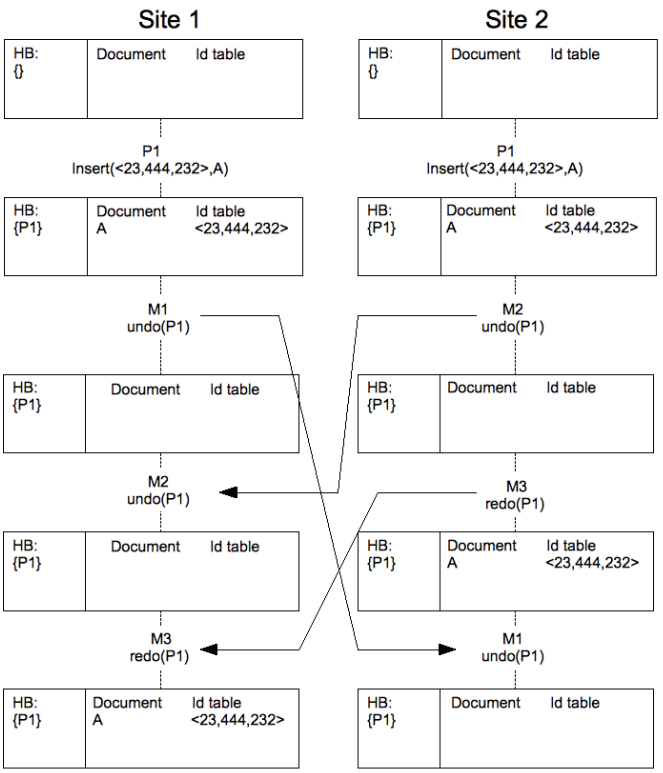

Fig. 3. Concurrent undo and redo messages counter-example

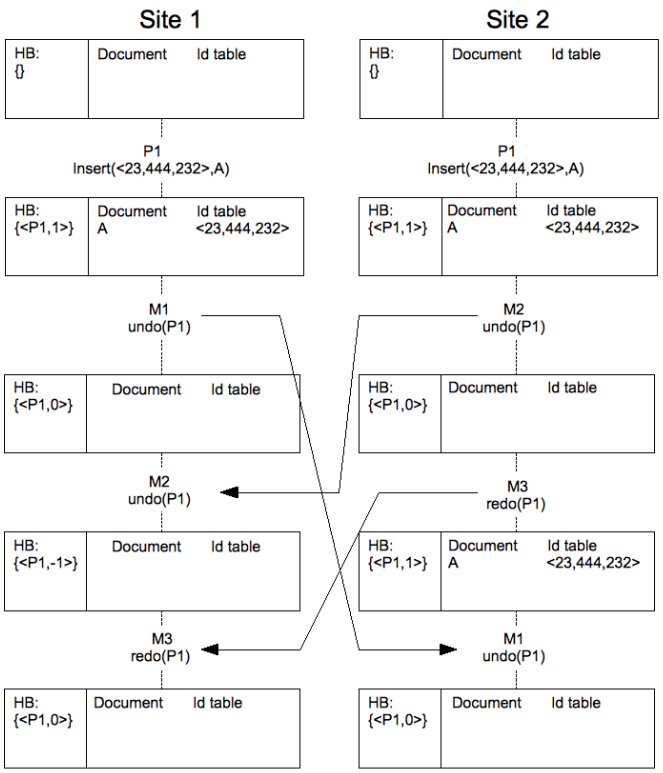

Fig. 4. Using patch degree to solve concurrent undo and redo scenario

```
    patch := HB.get(patchID)
    patch.degree−−;
    if (patch.degree = 0) then
      execute(inverse(patch))
    fi
  case Redo(patchID):
    patch := HB.get(patchID)
    patch.degree++;
    if (patch.degree = 1) then
      execute(patch)
    fi
  end;
end;
```

Thus, in the example Figure 4, Site1 will compute a degree of 0 and will not reapply $P1$.

5.3 Operation intention preservation

The patch degree allows managing concurrent undo/redo of the same patch, but lines can be deleted/reinserted concurrently by different patches.

For instance, let's assume that two sites host the same content (Figure 5). Concurrently, the first site removes the third line and adds a line at the end while Site 2 removes all lines from the document. Therefore, the third line was deleted by both sites. Finally, Site 1 undoes Site 2's modification. According to the undo intention preservation, the correct result is "ABD", i.e., the state before the reception of $P2$.

With the execute function of Section 4.4, every insert operation has an effect. When inverse($P2$), which contains the insertion of 'C', is applied, the line 'C' appears in the document. However, according to intention preservation, when Site2's modification is undone, Site1 deletion is the only modification effective, hence, the 'C' line must not appear in the document.

Thus, to know whether a line must appear or not in the document, we must know how many delete operations are undone and redone. For instance, in Figure 5, two delete operations are actives concerning the third line. Therefore, when Site 2 's deletion is undone, there is still one delete operation active, hence, we do not reinsert the third line.

We propose a solution based on the computation of the *line visibility degree*. When a line is created, it have a visibility degree of 1. Each time the line is deleted, we decrease its visibility degree. Similarly, when a delete is undone (or an insert redone), we increase the line visibility degree. A line with a visibility degree of 1 is said *visible* and appears in the document. Otherwise, the line is said *invisible* and does not appear in the document. For instance, on Site 1 of Figure 5, line 'C' have a visibility degree of

- 1 initially
- 0 after executing the patch $P1$
- -1 after executing the patch $P2$
- 0 after executing Undo($P2$)

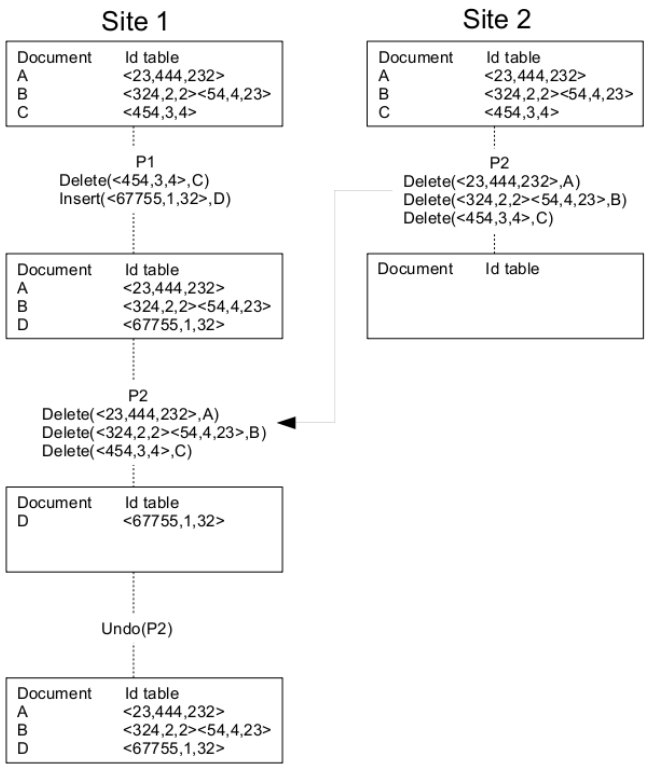

Fig. 5. Undo intention preservation

To compute the visibility degree of a line, different solutions can be built. A naive solution consists in walking the history buffer and counting operations in the active patches. Such an algorithm does not require additional storage but has a complexity proportional to the number of operations ever produced : $O(N)$. Another solution is the usage of tombstones. However, such tombstones do not need to be included in the document (contrary to the other tombstone-based approaches). Indeed, we use a special data structure called *cemetery*.

5.4 Cemetery

The cemetery is a hash table whose keys are line identifiers and values are visibility degrees. Fortunately most of the lines inserted in the document are not present in the cemetery :

- each line present in the document has a visibility degree of 1, hence, we do not store degrees for lines present in the document,
- lines with a visibility degree of 0 are not present in the cemetery.

Therefore, the cemetery contains only pairs $(id, degree)$ where the $degree$ is strictly less than 0, i.e., pairs for lines which have been deleted concurrently. As a result, we expect the cemetery memory overhead to be very low in systems with few concurrent editions.

The cemetery has two main methods:

- $cemetery$.get(id): returns 0 if the id is not in the cemetery otherwise it returns the visibility degree associated to this id,
- $cemetery$.set($id, degree$): inserts the pair $(id, degree)$ in the cemetery. If the id is already present, the old value is overwritten. If $degree = 0$, the pair is removed from the cemetery.

The execute function presented below is in charge of managing the visibility of the lines. The function applies the effect of an insert operation when the degree of the line becomes 1 and respectively it applies the effect of a delete operation when the degree becomes 0.

```
function execute(patch):
  for op in patch do
    switch (op)
    case Insert(id, content):
        degree := cemetery.get(id) + 1;
        if (degree = 1) then
          position := idTable.binarySearch(id);
          document.insert(position, content);
          idTable.insert(position, id);
        else
          cemetery.set(id, degree);
        fi
    case Delete(id, content):
        position := idTable.binarySearch(id);
        if (IdTable[position] = id) then
          document.remove(position, content);
          idTable.remove(position, id);
          degree := 0;
        else
          degree := cemetery.get(id) − 1;
        fi
        cemetery.set(id, degree)
    end;
  done
end;
```

Finally, Figure 6 illustrates the behavior of the cemetery.

6 Correctness of the Approach

Since the correctness of CES systems rely on the CCI criteria, we have to verify that Logoot-Undo respects causality, eventual consistency and intention preservation.

6.1 Causality

To ensure causality, we can use a scalable causal broadcast such as [20]. We could also use a probabilistic causal broadcast [36] in association with causal barriers [37], whose size is lower than vector clocks.

However, contrary to other OT or CRDT approaches, Logoot-Undo does not require causality to ensure eventual consistency. The reason is that every couples of operations commute (not only concurrent ones). Even the insertion of a line and its deletion. Indeed, a line can be "deleted" (visibility degree becomes -1) before being "inserted" (visibility degree becomes 0).

Thus, according to the targeted application, causality preservation can be removed. Also, a bounded – and thus scalable – mechanism that offers a good ordering accuracy from the user's point of view such as plausible clock [38] can be used.

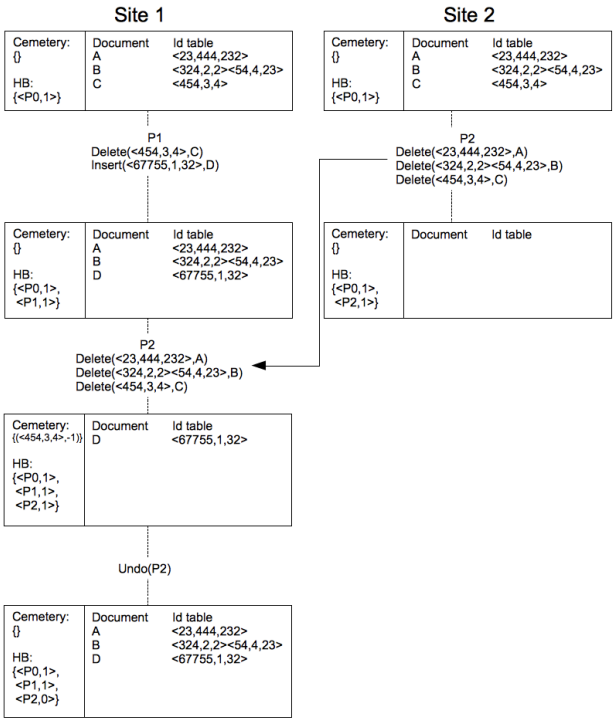

Fig. 6. Undo intention preservation with Cemetery

6.2 Eventual Consistency

To ensure convergence in the CRDT framework, the concurrent operations must commute. If line identifiers are unique, non mutable, and totally ordered, the different sites can apply any series of insert operations in any order and obtain the same result.

Lemma 1: Line identifiers are unique (i.e., there cannot be two different lines with the same identifier).

Proof: The last *position* of each line identifier contains the unique site number and the local clock of the site which generates the line. We assume that one site will not generate an identifier already in the document. Since, the pairs $(sid, clock)$ are unique, line identifiers are unique. □

Lemma 2: When all sites have received all modifications, all sites have the same history buffer configuration.

Proof: On each site, we get that the degree of a patch is equal to 1 minus the number of received Undo of this patch plus the number of received Redo. Since we assume that every message is eventually deliver once to all sites, all sites have the same set of patches with each the same degree, and thus the same HB configuration. □

Lemma 3: For one history buffer configuration, a line has one visibility degree.

Proof: Let's assume that a line with identifier id is inserted by a patch P_0. Let's call $P_1, ..., P_n$ all patches that deletes the line identified by id. The visibility value deg is:

$$deg(id, P_0, \ldots, P_n) = enable(P_0) - \left(\sum_{i=1}^{n} enable(P_i) \right) \quad (1)$$

with

$$enable(P_j) = \begin{cases} 0 & \text{if } P_j.degree < 1 \\ 1 & \text{else} \end{cases}$$

As a result, the visibility degree of a line depends only on the history buffer configuration. □

Theorem 4: Logoot-Undo ensures eventual consistency.

Proof: Since all modifications are eventually received, all sites have the same history buffer and the same visibility degree for each line. Thus each site has the same cemetery.

By definition, the identifiers table contains only identifiers with a visibility degree of 1. Since identifiers are unique and totally ordered, all sites have the same identifiers table. Finally, since one identifier corresponds to only one content, all sites have the same document. □

6.3 Intention preservation

Lemma 5: Logoot-Undo respects delete operation intention.

Proof: Equation (1) shows that the degree of a line can be at most 0 when a patch containing a delete operation of that line is enabled. Finally, with such degree, the line does not appear in the document. □

Lemma 6: Logoot-Undo respects insert operation intention.

Proof: Since positions are non-mutable and totally ordered, the order of lines observed on the generation replica will be preserved on other replicas. □

Lemma 7: Logoot-Undo respects undo intention.

Proof: Starting from the same assumptions as the proof of lemma 3, assume that a patch P_j is undone. As a consequence, we have Enable$(P_j) = 0$ by definition. According to Equation (1), we can directly write:

$$deg(id, P_0, \ldots, P_n) = deg(id, P_0, \ldots, P_{j-1}, P_{j+1}, \ldots, P_n)$$

As a result, all lines have the degree they would have if P_j was never executed. Since the document contains all lines whose degree is 1, the document returns in the state it would have reached if P_j was never produced. □

Theorem 8: Logoot-Undo respects intention preservation.

Proof: According to lemmas 5, 6 and 7, Logoot-Undo ensures the intention of all considered operations. As a result, Logoot-Undo ensures intention preservation. □

6.4 Complexity analysis

When building a line identifier of size k, the generation algorithm looks for the smallest $index$ value. For each increment, we compute an interval. Thus, assuming that the interval is computed incrementally, the time complexity of the generation algorithm is $O(k)$.

The integration algorithm uses the binary search algorithm to find a position. Therefore, this algorithm makes at most $log(n)$ comparisons, where n is number of lines in the document, and each comparison costs $O(k)$. Therefore, its time complexity is $O(k.log(n))$.

The size of line identifiers also affects the space complexity of the model storage which is $O(k.n)$: one identifier per line.

The tombstones-based approaches (WOOT and MOT2) have a complexity impacted by the presence of tombstones in the document. Indeed, to find an identifier or a position they must walk the model. Thus, they have a complexity relative to N: the number of elements in the model. N is equal to n plus the number of tombstones.

	Generation	Insertion	Deletion	Model
Logoot-Undo	$O(k)$	$O(k.log(n))$	$O(k.log(n))$	$O(k.n)$
WOOT	$O(N)$	$O(N^2)$	$O(N)$	$O(N)$
MOT2	$O(N)$	$O(N)$	$O(N)$	$O(N)$

k, the size of line identifiers is unbounded. Theoretically, the size of line identifiers can grow each time a line is inserted. Therefore, in the worst case, we have $k = N$. Thus, if no line is ever removed, the size of the document model overhead can be $\sum^N i = O(N^2)$.

[4] shows that k remains low without considering undo actions. Now, we have to verify that the novel strategy and the undo feature do not increase k in an unacceptable rate.

7 Evaluation

In this section, we present the experimentation made to compare Logoot-Undo to existing approaches. These experimentations are conducted to verify the following hypothesis:

- The Boundary Strategy (see Section 4.3) gives better results than the Random Strategy,
- Logoot-Undo performances – impacted by k value – scales better than tombstone-based approaches even in case of undo.

This experimentation was conducted, first, on a set of "extreme" pages to stress our algorithm scalability in presence of a great number of edits or lines, second, on a set of "regular" wiki pages.

7.1 Methodology

In order to verify our hypothesis, we replay the modifications applied to some Wikipedia pages in different collaborative editing systems.

To replay Wikipedia pages histories, we use the MediaWiki API [3] to obtain an XML file containing the revisions of a specific Wikipedia page. Then, using a diff algorithm [39], we compute the modifications performed between two revisions. These modifications are simply re-executed in our model. Since our approach generates each identifier randomly, we re-executed ten times each page history to obtain average values.

The only considered undo actions are reverts. A revert consists in restoring an old version, i.e., undoing all modifications made after this version. Reverts are detected by comparing the content of the wiki pages with the 10th previous revisions. For each pages, we show the results with and without the undo feature.

We measure the overhead for WOOT [32], [33], MOT2 [29] and Logoot-Undo.

Concerning Logoot-Undo, we make the experiments for the Random Strategy [4] and for the Boundary Strategy (Section 4.3).We arbitrary set the $Boundary$ to 1 million. In our implementation, we use 8-bytes integers, hence, a position contains 20 bytes (one integer, the site identifier and 4 bytes for a logical clock).

Concerning MOT2 [29], in order to support the undo feature, we use it combined with the TTF transformation functions [31] extended for undo support [30].

The overhead of WOOT and MOT2 are directly computed from the number and the type of operations performed on the document. Indeed, WOOT adds one "id" per inserted line, hence, its overhead is directly proportional to the number of insert operations. MOT2 simply replaces deleted lines by a tombstone, therefore, its overhead is directly proportional to the number of delete operations.

We have considered four categories of pages.

The most edited pages:[4] These pages are not encyclopedic articles. They can be discussion pages, forums or special pages mostly edited by bots. Such pages contribute widely in Wikipedia articles quality since they offer communication and coordination between Wikipedia users. In such pages, there is a lot of editions and the number of reverts highly depends on the considered page. For instance, in the page 1, around 18% of edits are reverts while in the page 7, reverts represent less than 0.002% of edits.

3. http://www.mediawiki.org/wiki/API:Query_-_Properties#revisions_.2F_rv

4. http://en.wikipedia.org/wiki/Wikipedia:Most_frequently_edited_pages

Id	Pages	Patches	Reverts
1	Wikipedia:Administrator_ intervention_against_vandalism	438330	79514
2	Wikipedia:Administrators'_ noticeboard/Incidents	343406	4803
3	Wikipedia:Sandbox	170440	25718
4	Wikipedia:Reference_ desk/Science	142722	851
5	Wikipedia:Reference_ desk/Miscellaneous	148283	1633
6	User:Cyde/List_of_candidates_ for_speedy_deletion/Subpage	152974	275
7	Wikipedia:WikiProject_Spam_ LinkReports	149538	2
8	Wikipedia:Help_desk	126509	1587
9	Wikipedia:Administrators'_ noticeboard	138460	2281
10	Template_talk:Did_you_know	91739	459

The most edited articles:[5] These pages are the most edited regular Wikipedia pages. In such pages, due to vandalism and "edit warring", reverts occupy a wide part of all edits. Indeed, in the considered pages, between 11% and 37% of edits are revert actions.

Id	Pages	Patches	Reverts
11	George_W._Bush	41563	14449
12	Wikipedia	28977	10953
13	United_States	24781	5770
14	Jesus	20271	4968
15	Wii	19991	4088
16	World_War_II	19547	4373
17	Adolf_Hitler	19489	5978
18	Britney_Spears	19244	3902
19	Deaths_in_2008	19027	2148
20	Michael_Jackson	18956	3735

The longest articles:[6]: The longest articles of Wikipedia are often lists of items. The ranking below is subject to important changes due to Wikipedia splitting policy. Indeed, for readability concerns, users are advised to split longest articles into several parts. These pages are poorly edited and contains few reverts.

Id	Page	Patches	Revert
21	Line_of_succession_to _the_ British_throne	3317	587
22	List_of_World_War _I_flying_aces	640	18
23	Timeline_of_United _States_inventions	3199	129
24	United_States_at_the_2008_Summer_Olympics	2314	50
25	List_of_college_athletic_programs_by_U.S. _State	868	6
26	List_of_Brazilian_football _transfers_2008	752	4
27	Licensed_and_localized_editions_of_Monopoly	318	9
28	Austrian_legislative _election,_2008	1086	19
29	List_of_sportspeople _by_nickname	2332	48
30	List_of_mass_murderers_and_spree_killers_by _number_of_victims	1172	25

Random articles: We have firstly selected 1000 random articles[7]. Due to the presence of "stubs", these pages are in average poorly edited. We have also randomly selected 1000 articles from the Wikipedia categories "Good Articles" and "Featured Articles"[8]. These articles meet quality criteria defined by the Wikipedia's editors. We choose these categories of articles since they meet the Wikipedia encyclopedic goal. Thus, they represent the aim of every Wikipedia article. In average, "Featured Articles" contain more edits than "Good Articles" which contain more edits than completely random ones. These data already show that a lots of edits are necessary to achieve encyclopedia goals.

Category	Nb of pages	Avg nb of Patches	Avg nb of Revert
Random Articles	1000	59.5	10.3
Good Articles	734	772.3	125.5
Featured Articles	266	1564.2	292.6
Total (GA+FA)	1000	982.9	169.9

For these four categories, we are interesting in the following values:

- Identifier size: For each category, we compute the average – over the last 100 editions – size of line identifiers.
- Memory overhead: For each of the treated pages, we present the average – over the last 100 editions – relative overhead of Logoot-Undo, WOOT and MOT2 approaches.

7.2 Results

Since the algorithmic complexity of Logoot depends on k – i.e., the average length of the line identifiers – this value must be as low as possible. Figure 7 shows that $k \leq 1.5$ for both strategy and for every page category except most edited pages. For the most edited pages, the Random Strategy obtains poor results in case of undo $k = 624.9$ while Boundary Strategy generates only positions of size 3.4 in average.

The following subsections presents the detailed comparison between the different approaches for each category. The presented overhead is the extra memory required to use these approaches. Values are presented relative in percentage. I.e., an overhead of 400 means 4 times (400%) the average size of the document.

7.2.1 Most edited pages

Figure 8 shows that the Random Strategy of Logoot-Undo gives acceptable results without the undo feature. Unfortunately, the introduction of the undo feature degrades Logoot-Undo performances and the overhead reaches very high values such as 255 times the document size. However, it remains smaller than the huge overheads of WOOT or MOT2. On the contrary, the Boundary Strategy generates the smallest overhead in average.

Figure 9 shows the overhead on a logarithmic scale of all considered approaches with undo. Except for the page 10, the Boundary Strategy provides the smallest overhead.

7.2.2 Most edited encyclopedic pages

Concerning the most edited articles (see Figure 10), Logoot-Undo has a very low overhead of around 10% with both strategies. Boundary Strategy generates a

5. http://en.wikipedia.org/wiki/Wikipedia:Most_frequently_edited_pages, namespace 0.

6. http://en.Wikipedia.org/wiki/Special:LongPages

7. Using http://en.wikipedia.org/wiki/Special:Random

8. See http://en.wikipedia.org/wiki/Good_Articles and http://en.wikipedia.org/wiki/Wikipedia:Featured_articles

Logoot-Undo	Most edited pages		Most edited articles		Longest articles		Good articles + Featured articles		Random articles	
	no undo	undo	no undo	undo	no undo	undo	no undo	undo	no undo	undo
Random Strategy	12.2	624.9	1.0	1.0	1.3	1.3	1.0	1.0	1.0	1.0
Boundary Strategy	3.0	3.4	1.0	1.0	1.3	1.5	1.0	1.0	1.0	1.0

Fig. 7. Average number of positions per line identifier.

Page	Logoot-Undo				Tombstone based approaches					
	Random Strategy		Boundary Strategy		Nb of tombstones		WOOT		MOT2	
	no undo	undo	no undo	undo	no undo	undo	no undo	undo	no undo	undo
1	27.8	27.8	27.8	27.8	425738	250982	359412.5	211888.6	17969.2	10593.0
2	27.2	177.9	13.4	13.8	1509768	1180704	1311.3	989.8	65.0	48.9
3	20.2	20.2	20.2	20.2	3320362	1905082	3175947.3	1822470.5	158796.4	91122.5
4	186.6	12621.9	8.9	8.9	408492	397963	4313.7	4202.8	215.2	209.7
5	520.4	16847.3	10.3	10.3	623084	520094	9365.4	7819.0	467.8	390.4
6	27.7	27.7	27.8	27.8	567810	566510	278281.1	277643.9	13912.7	13880.8
7	121.4	148.4	11.8	11.8	187627	187623	33955.4	33954.6	1697.2	1697.1
8	57.6	25575.3	13.0	13.0	557504	501821	11583.0	10426.0	578.5	520.7
9	37.9	10270.4	9.6	9.8	557944	457180	10695.8	8765.2	534.4	437.8
10	59.4	89.2	177.3	264.9	149579	147712	1906.1	1882.4	94.7	93.6
Average	108.6	6580.6	32.0	40.8	830791	611567	388677.2	238004.3	19433.1	11899.5

Fig. 8. Relative overhead for the most edited wiki pages

Page	Logoot-Undo				Tombstone based approaches					
	Random Strategy		Boundary Strategy		Nb of tombstones		WOOT		MOT2	
	no undo	undo	no undo	undo	no undo	undo	no undo	undo	no undo	undo
11	8.3	8.3	8.3	8.9	1618447	1228353	24318.0	18458.6	1215.5	922.5
12	15.3	15.3	15.3	15.3	1355418	1097374	30894.5	25016.5	1544.0	1250.1
13	8.3	8.3	8.3	8.3	580515	469363	7343.9	5939.3	366.8	296.5
14	9.8	9.8	9.8	9.8	328297	283128	5223.9	4505.4	260.7	224.8
15	8.7	8.7	8.7	8.7	170405	146142	4109.4	3524.8	205.0	175.8
16	8.3	8.3	8.3	8.3	354054	312515	5960.6	5262.3	297.6	262.7
17	8.5	8.5	8.5	8.5	350578	305561	3345.3	2916.6	166.8	145.3
18	9.0	9.0	9.0	9.6	190882	151621	4197.1	3335.5	209.4	166.3
19	16.6	17.4	16.6	33.0	43691	36820	1939.5	1635.9	96.1	81.0
20	7.1	7.1	7.1	7.1	198711	167676	3486.2	2946.0	174.0	146.9
Average	10.0	10.1	10.0	11.8	519100	419855	9081.8	7354.1	453.6	367.2

Fig. 10. Relative overhead for most edited articles

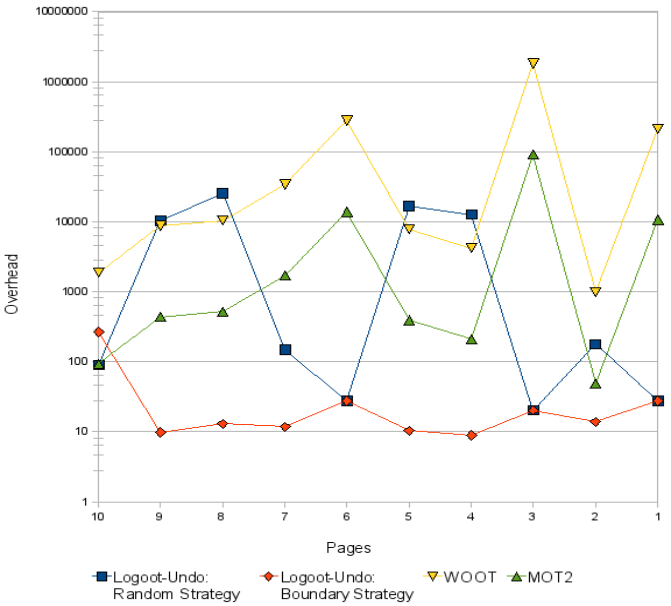

Fig. 9. Most Edited Pages

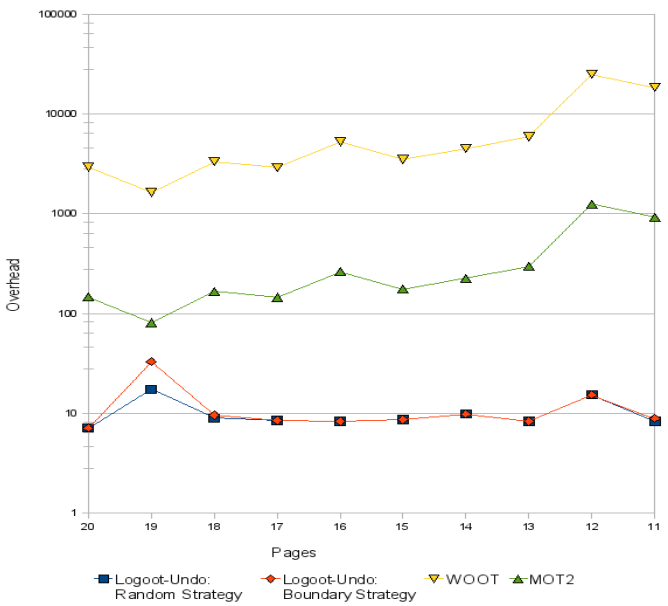

Fig. 11. Most Edited Articles

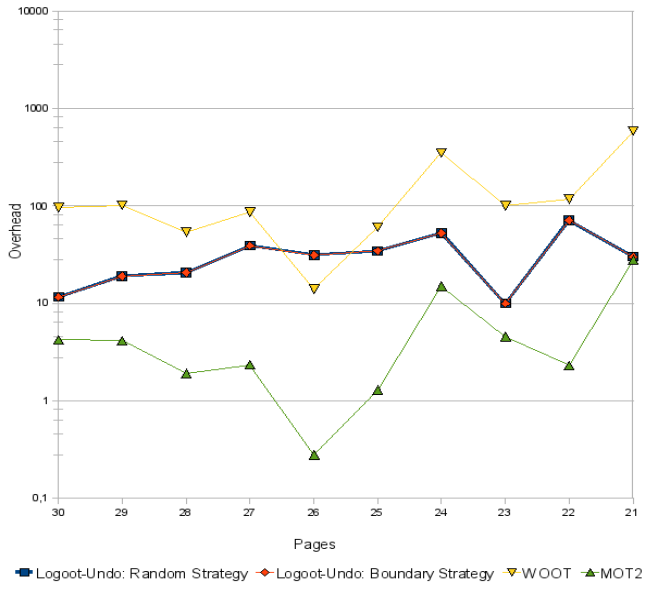

Fig. 13. Longest Articles

slightly worst overhead of 11.8% than Random Strategy which is 10.1% in case of undo. Compared to the overhead of WOOT and MOT2, Logoot-Undo overhead remains very low.

Finally, both strategies provide close results for Logoot-Undo. And, compared to WOOT and MOT2, Logoot-Undo achieves better overhead in any case. This is illustrated on Figure 11.

7.2.3 Longest articles

For the longest pages (see Figure 12), the Random Strategy performs better than the Boundary Strategy. However, their overheads remain close. Since these pages are poorly edited, tombstone based approaches have a small overhead. Since tombstones occupy less memory in MOT2 than in WOOT, MOT2 overhead is smaller. As a result, MOT2 has an average overhead of 6.4% in case of undo while WOOT overhead reaches 157%.

Figure 13 shows that Logoot-Undo overhead is between MOT2 and WOOT overheads.

7.2.4 Random Articles

Figure 14 shows the overhead obtained for the random pages. We notice that, for each approach, the overhead is higher for the "Featured Articles" than for the "Good Articles". For the tombstone-based approaches, the overhead is lower on completely random articles than in quality articles. On the contrary, the relative overhead of Logoot-Undo is higher for completely random articles and higher than MOT2 overhead.

For quality articles Logoot-Undo has the smallest overhead – around 11.5% – but MOT2 overhead is close – between 15% and 20%. Finally, WOOT overhead exceeds once the document size in case of undo and thrice with no undo.

7.3 Conclusions

Finally, from the whole experiment, we conclude that:

- As expected, the undo feature increases the overhead for Logoot-Undo. On the contrary, since the undo mechanisms of WOOT and MOT2 reuse tombstones, their overheads decrease while including undo modifications.
- Tombstone-based approaches eventually degrade in performance over time. Indeed, the number of edits on a document can only grow.
- The Boundary Strategy of Logoot-Undo has an overhead slightly bigger than the Random Strategy on most edited articles and on longest articles. Otherwise, on the most edited pages, the Boundary Strategy improves greatly Logoot-Undo overhead. Finally, in average, the Boundary Strategy offers the lowest overhead.
- Logoot-Undo with Boundary Strategy is the only approach that has for every category an overhead less than half of the document size. These experiments show that, even in case of undo, Logoot-Undo is more scalable than existing approaches.
- Thanks to its very small tombstones (one character per deleted line) and its absence of line identifier, MOT2 overhead is quite small in the average case. Unfortunately, its algorithmic complexity, which is proportional to the number of tombstones, will monotonically grow. Also, MOT2 dissemination uses a pair-wise reconciliation mechanism that exchanges messages whose size is proportional to the number of operations ever received on a peer. On the contrary, WOOT and Logoot dissemination mechanisms send only constant-size messages.

7.4 Limitations of the experiment

Since the Wikipedia uses a centralized wiki, we can expect a slightly different behavior in a P2P system.

Modifications in Wikipedia are serialized. Therefore, our experiments do not take account of concurrency. Since the cemetery size depends on the concurrency degree in the system, it remains empty during the experiments on Wikipedia. Fortunately, on collaborative editing system, this degree remains low: less than 3 edits per second on the whole English Wikipedia in average[9]. As a consequence, we expect the cemetery size to be low.

The experiments only include the "revert" feature which allow to return to a previous version. We do not include the general undo mechanism which allow to cancel any modification since the Wikipedia does not store such information.

9. 2.69 in October 2009 according to http://en.wikipedia.org/wiki/Wikipedia:Statistics

Page	Logoot-Undo				Tombstone based approaches					
	Random Strategy		Boundary Strategy		Nb of tombstones		WOOT		MOT2	
	no undo	undo	no undo	undo	no undo	undo	no undo	undo	no undo	undo
21	23.7	30.2	24.6	34.2	114511	110375	610.4	588.6	29.3	28.2
22	71.1	71.1	113.4	125.3	8856	8795	117.9	117.5	2.3	2.3
23	10.0	10.0	10.0	10.7	20686	19247	108.0	100.9	4.9	4.6
24	52.7	52.8	52.7	109.6	54057	47513	393.4	351.8	17.0	15.0
25	34.6	34.6	34.6	34.6	4198	4191	60.7	60.7	1.3	1.3
26	27.0	31.5	15.0	16.6	881	874	14.2	14.1	0.3	0.3
27	39.3	39.3	44.3	45.8	7303	7210	86.5	86.3	2.4	2.4
28	19.7	20.7	23.4	25.4	5252	4880	56.8	53.7	2.1	1.9
29	19.1	19.1	19.1	19.1	13255	13098	102.9	101.9	4.2	4.1
30	11.6	11.7	11.6	15.9	13612	12993	99.9	96.3	4.4	4.2
Average	30.9	32.1	34.9	43.7	24261	22918	165.1	157.2	6.8	6.4

Fig. 12. Relative overhead for the longest articles

Category Page	Logoot-Undo				Tombstones based approaches					
	Random Strategy		Boundary Strategy		Nb of tombstones		WOOT		MOT2	
	no undo	undo	no undo	undo	no undo	undo	no undo	undo	no undo	undo
Random Articles	31.7	31.7	31.7	31.7	420.6	284.6	177.6	106.8	11.1	6.9
Good Articles	11.0	11.0	11.0	11.0	4241.6	3469.1	249.9	112.2	14.9	12.2
Featured Articles	12.1	12.1	12.1	12.1	13657.9	10642.9	448.5	166.8	27.4	21.4
Average (GA+FA)	11.5	11.5	11.5	11.5	6746.3	5377.3	327.0	133.4	19.8	15.8

Fig. 14. Relative overhead for the random articles

8 Conclusion

We presented the Logoot-Undo algorithm. This algorithm is designed for handling highly dynamic content on structured or unstructured P2P network. It ensures the CCI consistency model and tolerates an unknown number of peer, a high degree of churn and network failures. This algorithm belongs to the class of CRDT algorithms. Compared to previous works, it integrates an "undo anywhere, anytime" feature required for distributed collaborative systems.

We proved the correctness of our approach and we measured the overhead of the undo feature and the performances of the algorithm. Even if the overhead of this CRDT algorithm increases with the undo feature, it remains low on a corpus of data extracted from Wikipedia.

Experiments are currently limited by the lack of corpus that includes concurrency. We are currently working on the building of a corpus that integrates concurrency. Next, we will be able to rank all CRDT algorithms and related optimistic replication algorithms based on the same corpus.

The Logoot-Undo algorithm is simple enough to be integrated with legacy software. The "Identifiers table", the "Cemetery" and the "History Buffer" can be integrated on top of existing documents. We are currently working on the development of two different approaches:

1) One with a structured P2P approach called Uniwiki [18] where the Logoot-Undo is integrated in each DHT nodes,
2) The other with an unstructured P2P network [17] where the Logoot-Undo algorithm is integrated in a distributed semantic wiki.

CRDT algorithms is a new class of algorithms. We proposed a commutative replicated ordered list data type, we are working on a commutative replicated ordered tree data type.

References

[1] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335–371, 2004.

[2] N. Preguiça, J. M. Marquès, M. Shapiro, and M. Letia, "A commutative replicated data type for cooperative editing," in *ICDCS*. Montreal, Quebec, Canada: IEEE Computer Society, June 2009.

[3] G. Oster, P. Urso, P. Molli, and A. Imine, "Data Consistency for P2P Collaborative Editing," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*. Banff, Alberta, Canada: ACM Press, nov 2006, pp. 259–267.

[4] S. Weiss, P. Urso, and P. Molli, "Logoot : a Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks ," in *ICDCS*. Montréal, Québec, Canada: IEEE Computer Society, June 2009.

[5] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys*, vol. 37, no. 1, pp. 42–81, 2005.

[6] 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)*, vol. 5, no. 1, pp. 63–108, Mars 1998.

[7] M. Shapiro and N. Preguiça, "Designing a commutative replicated data type," INRIA, Rapport de recherche INRIA RR-6320, October 2007. [Online]. Available: http://hal.inria.fr/inria-00177693/fr/

[8] H.-G. Roh, J. Kim, and J. Lee, "How to design optimistic operations for peer-to-peer replication," in *JCIS*, 2006.

[9] C. Sun, "Undo any operation at any time in group editors." in *CSCW*, 2000, pp. 191–200.

[10] D. Spinellis and P. Louridas, "The collaborative organization of knowledge," *Commun. ACM*, vol. 51, no. 8, pp. 68–73, 2008.

[11] G. D. Abowd and A. J. Dix, "Giving undo attention." *Interacting with Computers*, vol. 4, no. 3, pp. 317–342, 1992.

[12] A. Prakash and M. J. Knister, "A framework for undoing actions in collaborative systems." *ACM Trans. Comput.-Hum. Interact.*, vol. 1, no. 4, pp. 295–330, 1994.

[13] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser, "An integrating, transformation-oriented approach to concurrency control and undo in group editors." in *CSCW*, 1996, pp. 288–297.

[14] J. Ferrié, N. Vidot, and M. Cart, "Concurrent undo operations in collaborative environments using operational transformation." in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, ODBASE 2004*, ser. Lecture Notes in Computer Science, vol. 3290. Springer, Novembre 2004, pp. 155–173.

[15] C. Sun, "Undo as concurrent inverse in group editors," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 9, no. 4, pp. 309–361, Décembre 2002.
[16] H. Skaf-Molli, C. Rahhal, and P. Molli, "Peer-to-peer semantic wikis," in *The 20th International Conference on Database and Expert Systems Applications - DEXA'09*, Linz, Austria, September 2009.
[17] C. Rahhal, H. Skaf-Molli, P. Molli, and S. Weiss, "Multi-synchronous collaborative semantic wikis," in *The 10th International Conference on Web Information Systems Engineering (WISE'09)*, Poznan, Poland, October 2009.
[18] G. Oster, P. Molli, S. Dumitriu, and R. Mondejar, "UniWiki: A Reliable and Scalable Peer-to-Peer System for Distributing Wiki Applications ," in *Collaborative Peer-to-Peer Systems (COPS'09)*, groningen, Netherlands, June 2009, p. 21.
[19] C. Sun, "Optional and responsive fine-grain locking in internet-based collaborative systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 9, pp. 994–1008, 2002.
[20] A. D. Kshemkalyani and M. Singhal, "Necessary and sufficient conditions on information for causal message ordering and their optimal implementation," *Distributed Computing*, vol. 11, no. 2, pp. 91–111, 1998.
[21] L. Lamport, "Time, clocks, and the ordering of events in a distributed system." *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.
[22] P. R. Johnson and R. H. Thomas, "RFC 677: Maintenance of duplicate databases," Janvier 1975, (Septembre 2005), `http://www.faqs.org/rfcs/rfc677.html`.
[23] R. Li and D. Li, "A new operational transformation framework for real-time group editors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 3, pp. 307–319, 2007.
[24] B. C. Neuman, "Scale in distributed systems," in *Readings in Distributed Computing Systems*. IEEE Computer Society Press, 1994, pp. 463–489.
[25] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems." in *SIGMOD Conference*, J. Clifford, B. G. Lindsay, and D. Maier, Eds. ACM Press, 1989, pp. 399–407.
[26] M. Suleiman, M. Cart, and J. Ferrié, "Serialization of concurrent operations in a distributed collaborative environment." in *GROUP*, 1997, pp. 435–445.
[27] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman, "Copies convergence in a distributed real-time collaborative environment." in *CSCW*, 2000, pp. 171–180.
[28] D. Sun and C. Sun, "Operation Context and Context-based Operational Transformation," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*. Banff, Alberta, Canada: ACM Press, Novembre 2006, pp. 279–288.
[29] M. Cart and J. Ferrié, "Asynchronous reconciliation based on operational transformation for p2p collaborative environments," in *CollaborateCom*, 2007.
[30] S. Weiss, P. Urso, and P. Molli, "An Undo Framework for P2P Collaborative Editing ," in *CollaborateCom*, Orlando, USA, November 2008.
[31] G. Oster, P. Urso, P. Molli, and A. Imine, "Tombstone transformation functions for ensuring consistency in collaborative editing systems," in *The Second International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*. Atlanta, Georgia, USA: IEEE Press, November 2006.
[32] S. Weiss, P. Urso, and P. Molli, " Wooki: a P2P Wiki-based Collaborative Writing Tool," in *Web Information Systems Engineering*. Nancy, France: Springer, December 2007.
[33] C. Rahhal, S. Weiss, H. Skaf-Molli, P. Urso, and P. Molli, "Undo in peer-to-peer semantic wikis," in *The 4th Workshop on Semantic Wikis: The Semantic Wiki Web at the 6th Annual European Semantic Web Conference (ESWC)*, Crete, Greece, June 2009.
[34] B. Du and E. A. Brewer, "Dtwiki: a disconnection and intermittency tolerant wiki," in *WWW*, 2008, pp. 945–952.
[35] J. C. Morris, "Distriwiki: : a distributed peer-to-peer wiki network," in *Int. Sym. Wikis*, 2007, pp. 69–74.
[36] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Trans. Comput. Syst.*, vol. 21, no. 4, pp. 341–374, 2003.
[37] R. Prakash, M. Raynal, and M. Singhal, "An adaptive causal ordering algorithm suited to mobile computing environments," *J. Parallel Distrib. Comput.*, vol. 41, no. 2, pp. 190–204, 1997.
[38] F. J. Torres-Rojas and M. Ahamad, "Plausible clocks: constant size logical clocks for distributed systems," *Distrib. Comput.*, vol. 12, no. 4, pp. 179–195, 1999.
[39] E. W. Myers, "An o(nd) difference algorithm and its variations," *Algorithmica*, vol. 1, no. 2, pp. 251–266, 1986.

Stéphane Weiss received the engineering degree from the ESSTIN engineering school and, the MS degree in computer science from the University of Nancy (France) in 2006. He is currently a PhD student at the University of Nancy. He is mainly interested in massive collaborative editing, group undo, and P2P systems.

Pascal Urso received the MS and the PhD degree in computer science from the University of Nice - Sophia Antipolis, in 1998 and 2002, respectively. Since 2004, he is an assistant professor at the University of Nancy. His research interests include distributed systems, data replication, collaborative systems, P2P computing and automated theorem proving.

Pascal Molli graduated from Nancy University (France) and received his Ph.D. in Computer Science from Nancy University in 1996. Since 1997, he is Associate Professor at University of Nancy. His research topic is mainly Computer Supported Cooperative Work, P2P and distributed systems and collaborative knowledge building. His current research topics are: Algorithms for distributed collaborative systems, privacy and security in distributed collaborative systems, and collaborative distributed systems for the Semantic Web.