An efficient collaborative editing algorithm supporting string-based operations

Xiao Lv*†, Fazhi He*, Weiwei Cai* Yuan Cheng*
* State Key Laboratory of Software Engineering
Wuhan University, Wuhan, P.R.China 430072
Email: fzhe@whu.edu.cn
† College of Computer Engineering
Naval University of Engineering, Wuhan, P.R.China 430033

***Abstract*—Recently, Commutative Replicated Data Type(CRDT) algorithms have been proposed and proved by many references to outperform traditional algorithms in real-time collaborative editing. Replicated Growable Array(RGA) has the best average performance among CRDT algorithms. However, RGA only supports character-based primitive operations. This paper proposes an efficient collaborative editing approach supporting string-based operations(RGA Supporting String). Firstly, RGASS is presented under string-wise architecture to preserve operation intentions of collaborative users. Secondly, the time complexity of RGASS has been analyzed in theory to be lower than that of RGA and the state of the art OT algorithm(ABTSO). Thirdly, the experiment evaluations show that the computative performance of RGASS is better than that of RGA and ABTSO. Therefore, RGASS is more adaptable to large-scale collaborative editing with higher performance than a representative class of CRDT and OT algorithms in publications.**

***Keywords*—Real-time collaborative editing;Concurrency control;Operational transformation(OT);Commutative replicated data type(CRDT);String-based operation.**

I. Introduction

Real-time collaborative editing systems support natural and harmonious human to human interactions [1], which match well with the "Human-Centered Computing(HCC)" of the ACM Computing Classification System(CCS)[1] [2], [3]. A fully-replicated architecture is necessary to be adopted for co-editing systems, but also brings a great challenge for the consistency maintenance [1], [4]. Collaborative editing algorithms have been applied in widely collaborative applications, e.g. Google Wave/Docs[2], 2D spreadsheets [5], 2D images [6], 3D digital media design systems [7] and CAD [8], [9].

In the theoretical aspect, since the first OT system developed by Ellis and Gibbs [1], OT has been a significant research subject and gets a lot of explorations [10]–[18]. Recently, another class of collaborative editing algorithms called CRDT have been proposed for consistency maintenace [19]–[25]. CRDT algorithms place all objects into abstract data types in a total order and guarantee eventual consistency.

With the development of big data and cloud computing, real-time collaborative editing systems increasingly tend to massive-scale collaboration. In a large scale of collaboration, string-based operations are common and essential operations like copy-paste and find-replace. In addition, the performance is a key factor of the success of massive-scale collaboration. CRDT algorithms have been proved by many references to outperform traditional algorithms [23], [25]. RGA has been confirmed to have the best average performance in a representative class of CRDT algorithms [23]. Unfortunately, it does not support string-based operations, which is not well suitable for large-scale collaborative editing.

To address the above challenges, an efficient real-time collaborative editing algorithm called RGASS is proposed. Firstly, RGASS preserves operation intentions and supports string-based operations. Secondly, the time complexity in theory is analyzed to be lower than that of RGA and ABTSO. Thirdly, the experiment results show that RGASS has better computative performance than that of RGA and ABTSO. Therefore, RGASS is well suitable for a wide-area of massive-scale collaborative applications.

II. Background and Related work

Since the first OT algorithm was developed by Ellis and Gibbs in 1989 [1], OT algorithms have been proposed for nearly three decades. Especially, a plethora of important achievements proposed by C.Sun, D.Li and scholars from INRIA have been increasingly adopted in industrial applications, such as Jupiter [11], Nice [12], IBM OpenCoWeb[3], CoWord[4].

There are some challenges in OT algorithms. The original challenge is the convergence. There are two general approaches to achieve convergence. One is to design transformation functions satisfying TP1/TP2 [10]. The other is to design the total ordered transformation path capable of avoiding TP1/TP2 [11]–[13]. Past research has found that it is relatively easy to design the total ordered transformation path. Another challenge is preserving operation intentions of collaborative users. The operation intention is first proposed by Sun. But it has not been rigorously defined and measured. The first theoretical framework for the operation intention has been reported in the literature [16]. Based on this framework, a family of algorithms have been established for collaborative text editing [14]–[16]. In addition, the scalability is a key challenge of OT algorithms. Following Lamport and Sun, most OT algorithms use State Vectors to detect concurrency and causality. State Vectors are

[1]http://dl.acm.org/ccs/ccs.cfm
[2]http://docs.google.com
[3]https://github.com/opencoweb/coweb#readme
[4]http://www.codoxware.com

costly if a large amount of users are collaborating. Therefore, OT approaches generally do not scale well.

Most published OT algorithms do not support string-based operations. GOT describes how to support string-based operations, but no published work shows how to achieve string-based operations. ABTS supports string-based operations, but the time complexity is $O(|H|^2)$ [15]. ABTSO improves the time complexity to $O(|H|)$ by merging operations according to the operation effects relation. ABTSO has the best performance in string-based OT algorithms in publications [14].

Recently, CRDT algorithms have been proposed and proved to outperform traditional algorithms. WOOT is proposed in the literature [19], the control procedure is quite complex and time-consuming. TreeDoc is presented in the literature [21], which models the document as a binary tree. When insertions are always executed at the end of the text, identifiers have unbounded lengths. LOGOOT is introduced in the literature [20]. A large amount of insertions in the same part lead to unbounded size of identifiers. RGA is proposed in the literature [22], remote operations perform in $O(1)$ time to find the character. RGA has been proved to have the best average performance in typical CRDT algorithms.

In a representative class of CRDT algorithms, except the literature [24], [25], most CRDT algorithms only support character-based operations. The literature [25] is based on WOOT, the time complexity of integrating remote insertions is $O(k^2)$, k is the number of concurrent insertions. With the increase of concurrent insertions, it costs more time. The literature [24] is based on LOGOOT, unique compressed identifiers reduce the memory consumption. However, similar to LOGOOT, how to make sure causality is not given.

III. THE PROPOSED APPROACH

A. The overall framework

The overall framework is shown in Fig.1. Every site maintains a view and a model. A model is composed of a hash table(HT) and a double-linked list(L_{model}). L_{model} links all nodes in a total order. Every node represents a string. HT stores original and splitting nodes. A view is composed of a double-linked list(L_{view}). L_{view} links visible nodes in L_{model}. A local operation needs to find the target node via L_{model}. A remote operation needs to find the effective node via HT and the effective position via L_{model}. Synchronization between a view and a model needs to make the effects of integrated updates appear in the view.

A node is denoted as $Node$, which is a eight-tuple $<key, flag, visible, content, prior, next, link, list>$. The key is the unique identifier for each $Node$. The $flag$ shows that whether $Node$ is splitting or non-splitting. The $visible$ is a boolean variable whose value is true or false. The $content$ is the inserted or deleted string. Two pointers $next$ and $prior$ are used for linking $Nodes$ in L_{model}. The pointer $link$ is used for the double chaining in HT. The $list$ is used for reserving the splitting sub-nodes by insertions or deletions.

B. Related definitions

Definition 1 ($Node\prec$). *Given any two $Nodes$, $Node_i$ and $Node_j$, if $Node_i\prec Node_j$, iff: the position of $Node_i$ is before the position of $Node_j$ in L_{model}.*

Definition 2 ($Node\prec$ is a total order). *Given any three $Nodes$, $Node_i$, $Node_j$ and $Node_k$, then $Node\prec$ is a total order, iff:*
(1) $Node_i\prec Node_j$, or $Node_j\prec Node_i$, or
(2) $Node_i\prec Node_j$, $Node_j\prec Node_k$, then $Node_i\prec Node_k$.

Definition 3 (Intention of an operation O). *Given an operation O, $Node$ is the object of an operation O, the intention of an operation O is the relative position of $Node$ in L_{model}.*

Definition 4 (Consistency of the operation intentions). *Given $\Sigma=\{Node_1, Node_2, ..., Node_m\}$, let m be the number of $Nodes$ in L_{model}. $L=\{L_1, L_2, ..., L_n\}$, for any L_i, L_j ϵ L, $i\neq j$, i, $j\epsilon\{1,2,...,n\}$. L_i and L_j double link m Nodes of Σ. L_1, L_2,...,L_n are maintained by n sites respectively. For any two nodes $Node_k$ and $Node_l$ in L_i, any two nodes $Node_i$ and $Node_j$ in L_j, $k\neq l$ and $i\neq j$, k, l, i, $j\epsilon\{1,2,...,m\}$. When all operations issued have been executed in n sites, consistency of the intention of operations is achieved iff: $Node_k \overset{L_i}{\prec} Node_l = Node_i \overset{L_j}{\prec} Node_j$.*

Definition 5 ($Identifier(ID)$ of $Node$). *ID of $Node$ is a five-tuple $< s, ssv, site, offset, len >$ where*
(1)s is the identifier of a session, a global increasing number.
(2)ssv is the sum of State Vector(SV) of an operation.
(3)$site$ is the unique identifier of the site.
(4)$offset$ is the length from the leftmost position of the current $Node$ to the leftmost position of the original $Node$.
(5)len is the length of the string in current $Node$.

Definition 6 ($ID_{Node}\prec$). *Given two Nodes $Node_i$ and $Node_j$, ID of two Nodes are respectively ID_{Node_i} and ID_{Node_j}, if $ID_{Node_i}\prec ID_{Node_j}$, iff:*
(1)$ID_{Node_i}[s]<ID_{Node_j}[s]$, or
(2)$ID_{Node_i}[s]=ID_{Node_j}[s]$, $ID_{Node_i}[ssv]<ID_{Node_j}[ssv]$, or
(3)$ID_{Node_i}[s]=ID_{Node_j}[s]$, $ID_{Node_i}[ssv]=ID_{Node_j}[ssv]$, $ID_{Node_i}[site]<ID_{Node_j}[site]$, or
(4)$ID_{Node_i}[s]=ID_{Node_j}[s]$, $ID_{Node_i}[ssv]=ID_{Node_j}[ssv]$, $ID_{Node_i}[site]=ID_{Node_j}[site]$, $ID_{Node_i}[offset]>ID_{Node_j}[offset]$.

C. Integrating local operations

RGASS handles the following local operations. (1)LocalInsert(int nindex, int pindex, string str, $ID\ key_{cur}$). The parameter nindex is used for finding the target node. The parameter pindex is used for finding the operation position. The parameter str is the inserted string, key_{cur} is ID of the inserted string. (2)LocalDelete(int nindex, int pindex, int len, $ID\ key_{cur}$). The parameter len is the length of the deleted string, the parameter key_{cur} is ID of the current deletion.

Algorithm 1 presents how to integrate a local insertion. The function FindNode is used for finding the target node(Org_Node)(line 2). Then, a new node(New_Node) is created for the inserted string(line 3). First, Org_Node is head of L_{model}, New_Node is inserted after head and links for New_Node are created(lines 4-5). Second, Org_Node is not head of L_{model}, it is necessary to find Org_Node, then New_Node is inserted after Org_Node or into Org_Node(lines 7-13). According to the operation position($pindex$), there are two cases. First, New_Node is inserted after Org_Node(lines 7-8). Second, New_Node is inserted into Org_Node and Org_Node is split into $FNode$ and $LNode$. Then, links for $FNode$, $LNode$ and New_Node

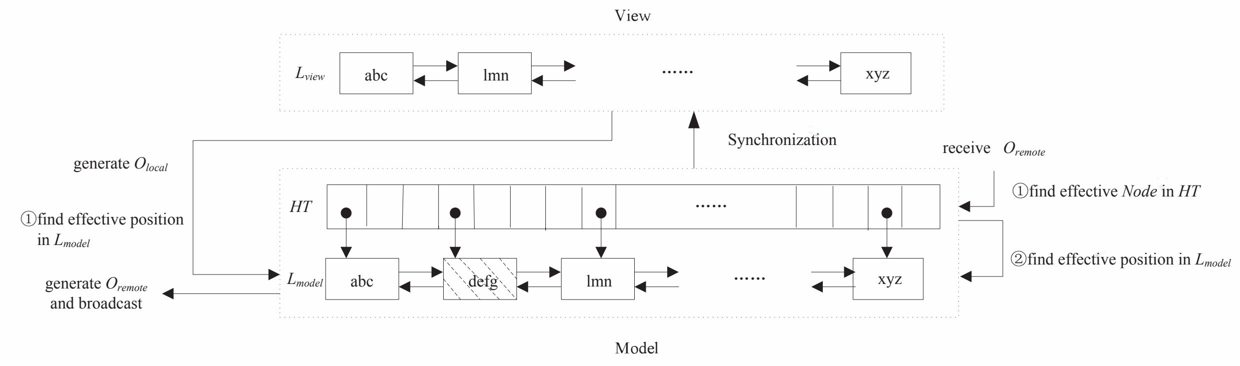
View
L_{view}
abc
lmn
......
xyz
Synchronization
receive O_{remote}
generate O_{local}
①find effective position in L_{model}
HT
①find effective Node in HT
②find effective position in L_{model}
L_{model}
abc
defg
lmn
xyz
generate O_{remote} and broadcast
Model

Fig. 1: The whole framework of RGASS.

are created and $FNode$ and $LNode$ are placed into HT(lines 10-12). At last, New_Node is placed into HT and a remote operation is generated and broadcast to other sites(lines 15-16).

Algorithm 1 The algorithm of integrating a local insertion.

```
1:  INPUT: nindex,pindex,str,$key_{cur}$
2:  $Org\_Node$=FindNode(nindex)
3:  $New\_Node$=new $Node$($key_{cur}$,str)
4:  if nindex==0 then
5:      double link $New\_Node$ next to head
6:  else
7:      if pindex==$Org\_Node.key$.len then
8:          double link $New\_Node$ next to $Org\_Node$
9:      else
10:         $Org\_Node$ is split into $FNode$ and $LNode$,
11:         create links for $FNode$, $New\_Node$ and $LNode$
12:         place $FNode$ and $LNode$ into $HT$
13:     end if
14: end if
15: place $New\_Node$ into $HT$
16: broadcast(pindex,$Org\_Node.key$,str,$key_{cur}$) to other
    sites
17: OUTPUT:$L_{model}$
```

Algorithm 2 presents how to integrate a local deletion. There are five cases. First, the function DeleteWholeNode is used for deleting the whole original node(Org_Node) and it is kept as a tombstone(lines 4-6). Second, the function DeletePriorNode is used for deleting the front part of Org_Node(lines 7-9). Org_Node is split into two sub-nodes and the first sub-node is kept as a tombstone. Third, the function DeleteLastNode is used for deleting the rear part of Org_Node(lines 10-12). Fourth, the function DeleteMiddleNode is used for deleting the middle part of Org_Node and the middle sub-node is kept as a tombstone(lines 13-15) . Fifth, multiple nodes are deleted(lines 16-18). The procedure is the combination of the above four cases.

D. Integrating remote operaions

RGASS handles the following remote operations. (1)RemoteInsert(int pindex, $ID\ key_{tar}$, string str, $ID\ key_{cur}$). The parameter key_{tar} is ID of the target node, which is used for

Algorithm 2 The algorithm of integrating a local deletion.

```
1:  INPUT: nindex,pindex,len,$key_{cur}$
2:  $ld$=new List$<ID>$()
3:  $Org\_Node$=FindNode(nindex),$l$=$Org\_Node.key$.len
4:  if pindex==1 and len==$l$ then
5:      $ld$.Add(DeleteWholeNode($Org\_Node$))
6:  end if
7:  if pindex==1 and len$<l$ then
8:      $ld$.Add(DeletePriorNode($Org\_Node$,len))
9:  end if
10: if pindex$>$1 and pindex+len-1==$l$ then
11:     $ld$.Add(DeleteLastNode($Org\_Node$,pindex-1))
12: end if
13: if pindex$>$1 and pindex+len-1$<l$ then
14:     $ld$.Add(DeleteMiddleNode($Org\_Node$,pindex,len))
15: end if
16: if pindex$>$1 and pindex+len-1$>l$ then
17:     $ld$.Add(DeleteMutipleNode($Org\_Node$,pindex,len))
18: end if
19: broadcast(pindex,len,$ld$,$key_{cur}$) to other sites
20: OUTPUT: $L_{model}$
```

finding the target node. (2)RemoteDelete(int pindex, int len, $ID\ key\_list$, $ID\ key_{cur}$). The parameter key_list is a list, which is used to reserve IDs of many deleted nodes.

Algorithm 3 gives how to integrate a remote insertion. The function FindEffectiveNode is used for finding the effective node($Node$)(line 2). When $Node$ is not head of L_{model}, there are two cases(lines 4-14). First, New_Node is inserted after $Node$(lines 5-9). Second, New_Node is inserted into $Node$, $Node$ is split into sub-nodes and links for New_Node, $FNode$ and $LNode$ are created(lines 11-13). When $Node$ is head of L_{model}(lines 16-22), some concurrent insertions may have already inserted their new nodes next to head. Therefore, it is necessary to scan the nodes next to head until a node whose key is first $\prec key_{cur}$(lines 17-20).

Algorithm 4 presents how to integrate a remote deletion. First, the deleted string involves only one node (lines 3-5). Second, the deleted string involves multiple nodes(lines 7-18). The function DelNode is used for deleting the splitting sub-nodes. A node may be split into two or three sub-nodes by concurrent operations. Therefore, the function DelNode needs to consider three cases. First, the deleted string(str) involves only one sub-node. Second, str involves two sub-nodes. Third, str involves three sub-nodes.

Algorithm 3 The algorithm of integrating a remote insertion.

```
INPUT: pindex,$key_{tar}$,str,$key_{cur}$
$Node$=FindEffectiveNode($key_{tar}$,pindex)
$New\_Node$=new $< Node >$($key_{cur}$, str)
if $Node$!=null then
    if pindex==$Node.key$.len then
        while ($Node$!=null and $key_{cur}$<$Node.key$) do
            $Node$=$Node$.next
        end while
        double link $New\_Node$ after $Node$
    else
        $Node$ is spilt into $FNode$ and $LNode$
        create links for $FNode$, $New\_Node$ and $LNode$
        place $FNode$,$LNode$ into $HT$
    end if
else
    initialize $pre$=head, $cur$=head.$next$
    while ($cur$!=null) do
        if $key_{cur}$<$cur.key$ then
            $pre$=$cur$,$cur$=$cur.next$
        end if
    end while
    double link $New\_Node$ after $pre$
end if
place $New\_Node$ into $HT$
OUTPUT: $L_{model}$
```

Algorithm 4 The algorithm of integrating a remote deletion.

```
INPUT: pindex,len,$key\_list$,$key_{cur}$
count=$key\_list$.count
if count==1 then
    $node$=$hash$($key\_list$[0])
    DelNode(pindex,len,$node$)
else
    $node$ = $hash$($key\_list$[0])
    DelNode(pindex,$node.key$.len-pindex+1,$node$)
    int sum_len=$node.key$.len-pindex+1
    int pos=1
    for (int i=1; i<count-1; i++) do
        $tempnode$=$hash$($key\_list$[$i$])
        DelNode(pos,$key\_list$[$i$].len, $tempnode$)
        sum_len+ =$key\_list$[$i$].len
    end for
    int lastlen=len-sum_len
    $lastnode$=$hash$($key\_list$[$count - 1$])
    DelNode(pos,lastlen,$lastnode$)
end if
OUTPUT: $L_{model}$
```

E. Synchronization between view and model

When a local(remote) operation is integrated, the updates of L_{model} need to be shown in L_{view}. Algorithm 5 presents the synchronization procedure.

Algorithm 5 The algorithm of synchronizing.

```
INPUT: $L_{model}$
$L_{view}$=new $List < Node >$()
$Node$ $node$=head.next
while ($node$!=null) do
    if $node$.IsVisible() then
        $L_{view}$.Add($node$)
    end if
    $node$=$node$.next
end while
OUTPUT: $L_{view}$
```

IV. Time complexity analysis and Experimental evaluation

A. Time complexity analysis

Table I shows the average time complexity of character operations for RGA, RGASS and ABTSO. Assume that the average number of nodes is N, the average number of characters in each node is d, the average splitting times of the target node is m and the average number of concurrent inserted nodes is n. H is an operation log, $H_i(H_d)$ is the insertion(deletion) log. $|H_i|(|H_d|)$ is the number of insertions(deletions). In RGASS, the best-case time complexity of a remote character insertion(deletion) is $O(1/d)$ as the target node is directly found by $hash$ function. The worst-case time complexity of a remote character insertion is $O((m+n)/d)$ in case that the target node has been split m times and the number of concurrent inserted nodes is n. The worse-case time complexity of a character remote deletion is $O(c \times 3^m/d)$, c is a constant. In this case, the target node has been split m times. In each split, the target node is split into three sub-nodes and the deleted string involves three sub-nodes. Obviously, RGASS overwhelms RGA and ABTSO, especially in the best-case remote insertions(deletions).

TABLE I: The average time complexity of character operations for RGA, RGASS and ABTSO.

Algorithms	Local operations		Remote operations	
	insertion	deletion	insertion	deletion
RGA	$O(N)$	$O(N)$	Best:$O(1)$ Worst:$O(n)$	$O(1)$
RGASS	$O(N/d)$	$O(N/d)$	Best:$O(1/d)$ Worst:$O((m+n)/d)$	Best:$O(1/d)$ Worst:$O(c \times 3^m/d)$
ABTSO	$O(\|H_d\|/d)$	$O(\|H_d\|/d)$	$O((\|H_i\|+\|H_d\|)/d)$	$O((\|H_i\|+\|H_d\|)/d)$

B. Experimental evaluation

In this section, we compare RGASS with previous RGA and ABTSO, three algorithms are all implemented in C# language and compiled by Visual Studio 2010 on windows 7 system, Intel Core i7, 3.60GHz CPU. We designed the real-time collaboration in order to obtain operation logs. Then, replay all operations on three algorithms. In our following experiments, we assume that two sites concurrently generate random string-based operations at random positions. Due to the page limit of this paper, the insertion ratio is set to 80%. We run three algorithms for 25 times in each operation log and the average values are recorded. First, we compare the computative time of RGASS with that of RGA. Second, we compare the computative time of RGASS with that of ABTSO.

(1) When each string-based operation includes fixed-size characters, with the number of operations increases, we present the performance behaviour over time for RGA and RGASS. Fig. 2 shows the computative time of local and remote character operations for RGA and RGASS. Apparently, RGASS has lower execution time than RGA. For example, when the number of operations is 3000, the execution time of local and remote character operations for RGA are respectively 1475μs and 0.5μs. But RGASS only takes 0.1μs and 0.02μs. More importantly, with the number of operations increases, the execution time of remote character operations for RGASS decreases.

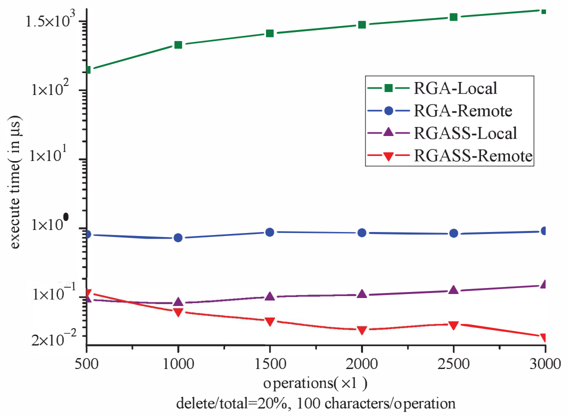

Fig. 2: Time of character operations of RGA and RGASS when the number of operations increases with step 500.

(2) When the number of operations is fixed, with the number of characters increases, we present the performance behaviour over time for RGA and RGASS. Fig. 3 shows the execution time of local and remote character operations for RGA and RGASS. Obviously, the execution time of RGASS is much lower than that of RGA. Especially, with the number of characters increases, the execution time of local(remote) character operations for RGASS decreases.

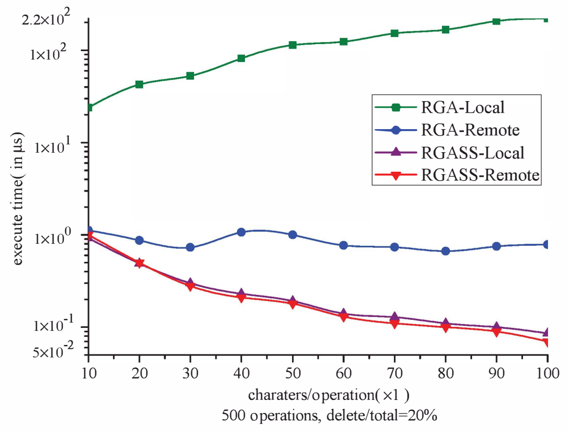

Fig. 3: Time of character operations of RGA and RGASS when the number of characters increases with step 10.

(3) When the number of remote operations and the length of H are relatively small, we study how long RGASS and ABTSO take to integrate N remote string-based operations into history H. Fig. 4 shows the computative time of remote operations for ABTSO and RGASS. When the length of H increases, ABTSO takes more time, but RGASS takes less time. For example, when the length of H is 200, ABTSO takes 68ms to integrate 100 remote operations and RGASS only takes 4ms.

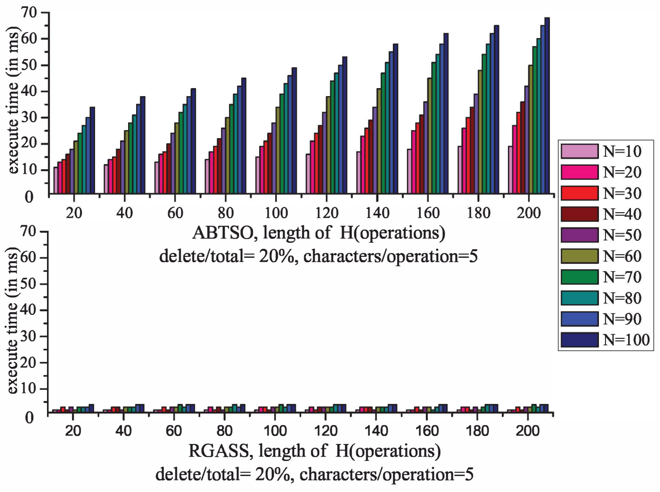

Fig. 4: Time to integrate N remote string-based operations of ABTSO and RGASS in a small scale of collaboration.

(4) When the number of remote operations and length of H increase dramatically, we study how long RGASS and ABTSO take to integrate N remote string-based operations into history H. Fig.5 shows the integration time of ABTSO and RGASS.

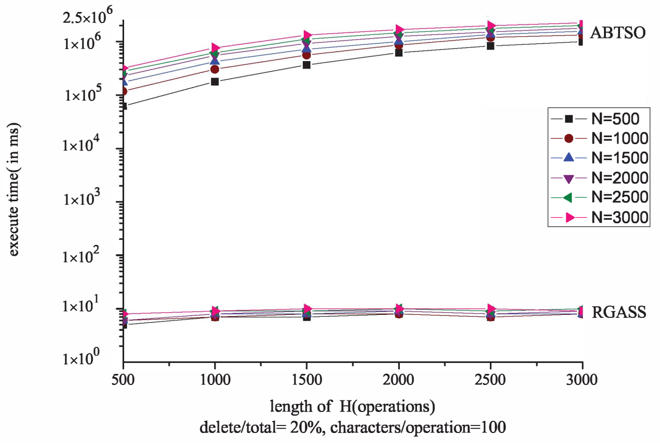

Fig. 5: Time to integrate N remote string-based operations of ABTSO and RGASS in a large scale of collaboration..

As shown in Fig.5, with the increase of the length of H, ABTSO takes long time to integrate a large amount of remote operations, but RGASS takes less time. The main reason is that every string-based deletion of ABTSO is processed as a sequence of character deletions. For example, when the length of H and the number of remote operations are both 3000, the delete ratio is 20%. In fact, ABTSO needs to integrate 2400 insertions and 600×100 character deletions. However, RGASS only needs to integrate 2400 insertions and 600 deletions, and it only takes around 5-10ms to integrate a large number of remote operations.

V. Conclusions and future work

The proposed algorithm preserves the intention of operations and achieves the eventual consistency. Theoretical analysis and experimental evaluation show that the computative performance of RGASS overwhelms RGA and ABTSO greatly. Therefore, RGASS is adaptable to massive-scale collaborative editing. In future work, we will extend the idea to collaborative CAD [26]–[31].

Acknowledgment

This research has been funded by the National Science Foundation of China (Grant No. 61472289 and 61502353) and the Natural Science Foundation of Hubei Province(Grant No. 2015CBF254).

References

[1] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," in *Acm Sigmod Record*, vol. 18, no. 2. ACM, 1989, pp. 399–407.

[2] R. Jain, "Eventweb: Developing a human-centered computing system," *Computer*, no. 2, pp. 42–50, 2008.

[3] B. Tomlinson, E. Blevis, B. Nardi, D. J. Patterson, M. Silberman, and Y. Pan, "Collapse informatics and practice: Theory, method, and design," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 20, no. 4, p. 24, 2013.

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

[5] C. Sun, H. Wen, and H. Fan, "Operational transformation for orthogonal conflict resolution in real-time collaborative 2d editing systems," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 2012, pp. 1391–1400.

[6] X. Wang, J. Bu, and C. Chen, "Achieving undo in bitmap-based collaborative graphics editing systems," in *Proceedings of the 2002 ACM conference on Computer supported cooperative work*. ACM, 2002, pp. 68–76.

[7] C. Sun *et al.*, "Dependency-conflict detection in real-time collaborative 3d design systems," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013, pp. 715–728.

[8] L. Gao, B. Shao, T. Lu, and N. Gu, "Maintaining semantic intention of step-wise operations in replicated cad environments," in *Computer Supported Cooperative Work in Design, 2008. CSCWD 2008. 12th International Conference on*. IEEE, 2008, pp. 154–159.

[9] Y. Cheng, F. He, and D. Zhang, "To support human-human interaction in collaborative feature-based cad systems," in *Human Centered Computing*. Springer, 2015, pp. 609–618.

[10] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser, "An integrating, transformation-oriented approach to concurrency control and undo in group editors," in *Proceedings of the 1996 ACM conference on Computer supported cooperative work*. ACM, 1996, pp. 288–297.

[11] 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*. ACM, 1995, pp. 111–120.

[12] H. Shen and C. Sun, "Flexible notification for collaborative systems," in *Proceedings of the 2002 ACM conference on Computer supported cooperative work*. ACM, 2002, pp. 77–86.

[13] Y. Xu, C. Sun, and M. Li, "Achieving convergence in operational transformation: conditions, mechanisms and systems," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 2014, pp. 505–518.

[14] B. Shao, D. Li, and N. Gu, "An optimized string transformation algorithm for real-time group editors," in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*. IEEE, 2009, pp. 376–383.

[15] B. Shao, D. Li, and N. Gu, "Abts: A transformation-based consistency control algorithm for wide-area collaborative applications," in *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*. IEEE, 2009, pp. 1–10.

[16] D. Li and R. Li, "An admissibility-based operational transformation framework for collaborative editing systems," *Computer Supported Cooperative Work (CSCW)*, vol. 19, no. 1, pp. 1–43, 2010.

[17] N. Gu, Q. Zhang, J. Yang, and W. Ye, "Dcv: a causality detection approach for large-scale dynamic collaboration environments," in *Proceedings of the 2007 international ACM conference on Supporting group work*. ACM, 2007, pp. 157–166.

[18] C. W. He Fazhi, Lv Xiao, "An efficient preserving intention operational transformation for real-time collaborative editing," *Chinese Journal of Computers*, vol. 38, no. 51, 2015.

[19] 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*. ACM, 2006, pp. 259–268.

[20] 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*. IEEE, 2009, pp. 404–412.

[21] N. Preguica, J. M. Marques, 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*. IEEE, 2009, pp. 395–403.

[22] 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*, vol. 71, no. 3, pp. 354–368, 2011.

[23] M. Ahmed-Nacer, C.-L. Ignat, G. Oster, H.-G. Roh, and P. Urso, "Evaluating crdts for real-time document editing," in *Proceedings of the 11th ACM symposium on Document engineering*. ACM, 2011, pp. 103–112.

[24] L. André, S. Martin, G. Oster, and C.-L. Ignat, "Supporting adaptable granularity of changes for massive-scale collaborative editing," in *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*. IEEE, 2013, pp. 50–59.

[25] W. Yu, L. André, and C.-L. Ignat, "A crdt supporting selective undo for collaborative text editing," in *Distributed Applications and Interoperable Systems*. Springer, 2015, pp. 193–206.

[26] F. He and S. Han, "A method and tool for human–human interaction and instant collaboration in cscw-based cad," *Computers in Industry*, vol. 57, no. 8, pp. 740–751, 2006.

[27] S. Jing, F. He, S. Han, X. Cai, and H. Liu, "A method for topological entity correspondence in a replicated collaborative cad system," *Computers in Industry*, vol. 60, no. 7, pp. 467–475, 2009.

[28] X. Cai, X. Li, F. He, S. Han, and X. Chen, "Flexible concurrency control for legacy cad to construct collaborative cad environment," *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, vol. 6, no. 3, pp. 324–339, 2012.

[29] Y. Cheng, F. He, X. Cai, and D. Zhang, "A group undo/redo method in 3d collaborative modeling systems with performance evaluation," *Journal of Network and Computer Applications*, vol. 36, no. 6, pp. 1512–1522, 2013.

[30] D. Zhang, F. He, S. Han, and X. Li, "Quantitative optimization of interoperability during feature-based data exchange," *Integrated Computer-Aided Engineering*, vol. 23, no. 1, pp. 31–51, 2016.

[31] Y. Cheng, F. He, Y. Wu, and D. Zhang, "Meta-operation conflict resolution for human–human interaction in collaborative feature-based cad systems," *Cluster Computing*,DOI:10.1007/s10586-016-0538-0, 2016.