Multi-core Accelerated Operational Transformation for Collaborative Editing

Weiwei Cai, Fazhi He(✉), and Xiao Lv

School of Computer, Wuhan University, Wuhan, China
fzhe@whu.edu.cn

Abstract. This article proposes a parallel operational transformation (OT) algorithm for collaborative editing. OT maintains the eventual consistency of replicated data in optimistic way, allowing users to manipulate the shared document simultaneously. It has been the first choice for most collaborative applications. However, existing approaches must keep the number of operations generated in a session small so that it can provide a decent responsive time. The multi-core/many-core architectures are becoming pervasive in recent years. Unfortunately, there is no prior work which has explored accelerating operational transformation algorithms with available computation power. We present a lock-free operation history which are accessed by a batch of remote operations at the same time. Moreover, a data parallel computation model is constructed to accelerate the integration of local operations. To the best of our knowledge, this is the first parallel OT algorithm. Experimental results show our proposed algorithm outperforms the stat-of-art algorithms for collaborative editing.

Keywords: Collaborative computing · Collaborative editing · Operational transformation · Parallel computing

1 Introduction

Collaborative editing systems constitute a class of collaborative computing systems where users modify the shared data separately and achieve a consistent result. To satisfy the requirement of high responsiveness and availability, these systems are based on data replication. Each user can freely edit any part of local copy and changes are immediately reflected on user interface. The generated updates are propagated to other users. The different execution order of these operations may lead to a divergence. Operational transformation (OT) maintains the consistency of replicas by changing the execution form of received operations. The technique has been applied in supporting various collaborative applications, such as Google Docs, Microsoft CoWord and CoPowerPoint [1], CoRED [2], SyncLD [3], and CoCAD [4].

OT transforms received operations with executed operations before executing them. Concurrent operations are commutative by transformations. As an example, consider the scenario where user1 inserts ‘a’ at position 1 and concurrently user2 deletes element at 2. Here, a document is represented as a string

S. Guo et al. (Eds.): CollaborateCom 2015, LNICST 163, pp. 121–128, 2016.
DOI: 10.1007/978-3-319-28910-6_11

of characters, "abc". When receiving remote operations, the deletion is transformed to operation at position 3 because user1 has inserted a character before its effect operation and the insertion does not change itself after transformation. As a result, both of the user1 and user2 obtain the consistent result "aab". All executed operations are recorded in history. The state of the art OT has a time complexity of $O(n)$ [5], where n is the number of editing operation in the history. Therefore, it is important to keep the operation history small for high responsiveness. With the widespread use of multi-core/many-core processors, the parallel computing power can efficiently address this limitation.

In this paper, we design a lock-free queue storing executed operations which allow several threads to process transformations in parallel. When there is an actual conflict between operations, some of them need to retry but never be blocked. Local operations are integrated individually but can still benefit from the parallel loop.

2 Related Work

The transformation function changes the operation form so that a pair of operations can execute out of order. It should satisfy the transformation properties (TP1 and TP2) [6]. TTF (tombstone transformation function) constructs a two-tier model with an extra layer which retains the deleted objects. It is correct based on the fact transformed operations do not consider the effect of delete operations. Imine et al. [7] contribute a set of correct transformation functions for the modified insertions. From another perspective, ABT [8] requires all insert operations must be located before delete operation in history. In terms of the basic theory, they are equivalent. Recently, Imine et al. [7] contribute a set of correct transformation functions for the modified insertions. The added attribute records the number of deletions which have been executed before the effect position.

The relationship between operations is traditionally determined by the state vector [9]. Based on this technique, algorithms perform poorly on separating concurrent operations from history [10]. WOOT algorithm [11] defined that a newly inserted element semantically depends on the previous and next element. It maintains the data model like TTF. The improved version WOOTH has a time complexity $O(n)$, where n is the number of inserted elements. As the asynchronous version of ABT, ABST [5] provides the transformation of long operation sequences and improved the time complexity to $O(H)$, where $|H|$ is the size of history.

3 Parallel Implementation

Operation history is both modified by local and remote threads. Therefore, they should be executed in mutual exclusive manner. In the following, we discuss the integration of local and remote operations respectively.

3.1 Data Model

Two tier data model is constructed as Fig. 1, where the logical view corresponds to the user view and the physical view retains the deleted objects. Both of them are modeled as a linear collection of elements. In logical view, the element can be various data type, such as characters, XML nodes. In physical view, each slot records the identifiers of operations which have inserted them, the number of visible element before it and the visible flag. Because an operation can uniquely be identified with the site identifier and local sequence number, it is easy to find its effect element.

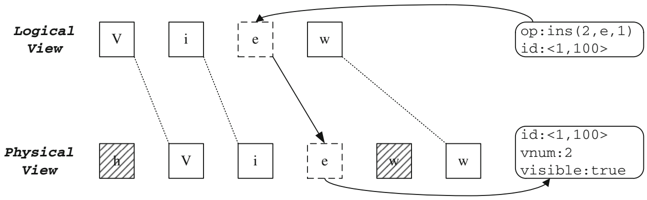


Fig. 1. Data model.

3.2 Operation Dependency Relation

The traditional dependency relations are based on the happened-before theory defined by Lamport [12]. If o_1 is executed before o_2, it is widely accepted that the precedence order should be preserved at all sites. As a matter of fact, there exist extensive cases where a pair of $o_1 \rightarrow o_2$ is able to execute out of order. For instance, assume that $o_1 \rightarrow o_2$ and $o_1 = ins(1, a)$, $o_2 = ins(3, b)$, then $[ins(1, a)\ ins(3, b)] = [ins(2, b)\ ins(1, a)]$. If two adjacent operations in history are not transformation-based commutative, the latter one is semantically dependent on the former one [13]. In other words, the boundary of causal and concurrent operation is naturally the dependent one.

However, this relation is determined by checking all operations executed before the target operation one by one, which is not easily parallelized. From the perspective of object positions relation, the operation of an element is only semantically dependent on operations which have manipulated its directly adjacent elements.

3.3 Integration of Local Operations

When users issue an operation, it is instantly executed and reflected on user interface. Then, the algorithm computes its corresponding position in physical view and sends the update to other users. Suppose the user executes an insertion $ins(p, e)$, the conversion has to find the $(p-1)$-th visible element in physical view. This procedure is generally implemented with a while-loop. However, it is not efficient to execute while loop in parallel due to the uncertain stop condition. To make it easier, each element records the number of visible elements before it. Now, the computation of position is a parallel for loop with the definite lower and upper bounds. After that, the dependency information becomes clear. As discussed above, the operation depends on the one which has inserted the previous or next object, or does not depend on any operation. The procedure of integrating insert operations is described in Algorithm 1. The same idea applies when integrating a delete operation.

Algorithm 1. The integration of insert operations

```
pos_l = op.pos;
if op.type = ins then
    #parallel for
    for i = 0;i < |P|;i++ do
        if P[i].vnum == pos_l − 1 then
            pos_p = i;
        else if P[i].vnum >= pos_l then
            P[i].vnum++;
        end if
    end for
    P.insert(pos_p, newEle);
end if
```

3.4 Integration of Remote Operations

We integrate a sequence of remote operations in parallel. Based on the fact that transforming o_1 against o_2 does not change o_2, each remote operation can safely be transformed with the operation history. Received operations are stored in a lock-free queue, denoted as *RQueue*, which can be processed by parallel thread. This concurrent containers can be found in some implementations, such as Intel's Threading building blocks[1] and open source library libcds[2]. Therefore, a batch of operations can be processed simultaneously in non-blocking way. If a pending operation satisfies the dependency relation, it is dequeued from *RQueue* and transformed with the operation history. Otherwise, it is appended to the *RQueue*. The operation history is also stored in a lock-free queue, denoted as *HQueue*, which only has the enqueue function. Because no operation node should be released, ABA problem does not exist [14].

Since the transformed operations are appended to the end of *HQueue*, only the *tail* node may be modified by simultaneous thread. Therefore, the observed operation history can be transformed with remote operations safely. Only when the thread reads *tail* node without interference, it enqueues the target operation. It needs the help of synchronized primitive *CAS* which is supported

[1] https://www.threadingbuildingblocks.org/.
[2] http://libcds.sourceforge.net/doc/cds-api/index.html.

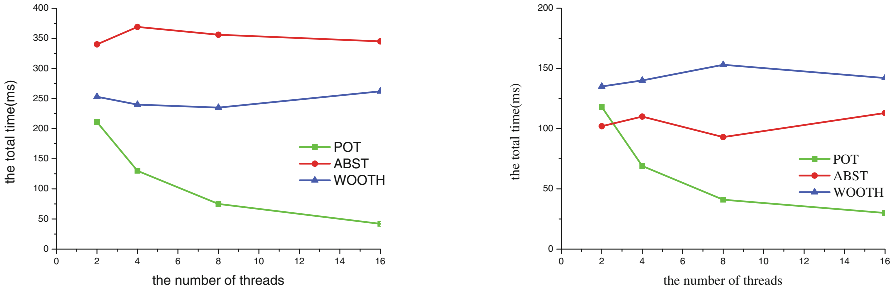
Algorithm 2. Integrate of remote operations

```
op = RQueue.dequeue();
cur = RQ.head, end = RQ.tail, newNode = newNode(op.pos, op.id);
if op.dep = null then
    LTransform(op, cur, end), newNode.pos = op.pos;
    while !CAS(RQ.tail, end, newNode) do
        end = RQ.tail;
        LTransform(op, cur, end); newNode.pos = op.pos;
    end while
else
    while cur! = end.next do
        if cur.id = op.id then
            while !CAS(RQ.tail, end, newNode) do
                end = RQ.tail;
                LTransform(op, cur, end), newNode.pos = op.pos;
            end while
            break;
        end if
        cur = cur.next
    end while
    RQ.enqueue(op);
end if
```

by most multiprocessor architectures. $CAS(reg, oldValue, newValue)$ compares the contents of a memory location (reg) to a given old value, only if they are the same, successfully modifies the contents of that memory location to a given new value. Algorithm 2 describes the integration of remote operations. $LTransform(op, cur, end)$ transforms op against operations from the current node to the end node, and finally cur and end point to the same location. If the *tail* node is modified during this period, the target operation continues to be transformed until the current node reach the end and then check the *tail* node again. After all remote operations is completed, they will sequentially update the physical view without extra computation.

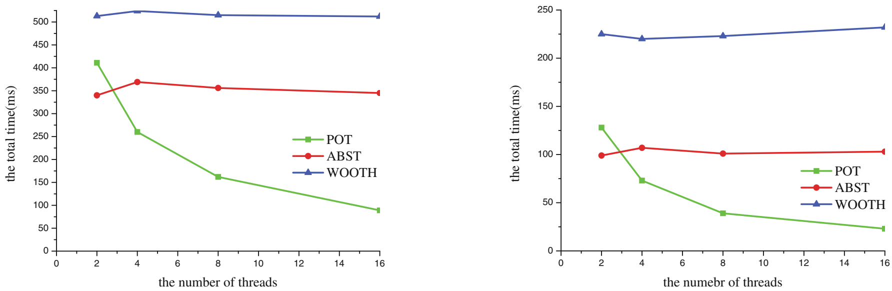
4 Evaluation

We evaluate the time consuming of local and remote operation with parallel OT algorithm (POT), and compare it with representative algorithms, WOOTH [15], and ABST [5]. All the algorithms were implemented in C++ and compiled with the same flags. Because ABST only maintains the operation history and WOOTH maintains the two tier data model like Fig. 1 but no need of the history, we simulate collaborative workloads on a non-empty document (with 10, 0000 characters) and empty document. By convention [15–17], we construct the operation history with 10, 0000 operations, where 80 % are insertions and positions are uniform distribution. Then we calculate the total time of integrating 100



(a) local operations (b) remote operations

Fig. 2. Empty document scenario.



(a) local operations (b) remote operations

Fig. 3. Non empty document scenario.

local and remote operations, respectively. The experiments were performed on a platform of two 4-core Intel i7-4770 with HyperThreading, 16 GB DDR3 RAM.

According to Fig. 2, ABST algorithm consumes more time in integrating local operations for the reason that it orders the history according to the effect position relation. However, when processing remote operations, ABST outperforms WOOTH and POT (with 2 threads). In Fig. 3, the number of elements in physical view is the double of that in the empty-document scenario, causing that the performance of WOOTH and POT both degrade, but it has little influence on ABST. When integrating remote operations, comparing with the empty-document scenario, only WOOTH slightly increases the time cost. POT acquires great improvement with more parallel thread in both scenarios. As a whole, POT outperforms the WOOTH and ABST.

5 Conclusion

In this paper, we contribute a parallel OT algorithm (POT) for collaborative editing. The proposed method accelerates the integration of local and remote operations with the support of multi-core architecture. We construct a two tier data model which helps compute the position in physical view and dependency relation of local operations in parallel. To remote updates, a lock-free queue storing executed operations can be accessed by simultaneous threads. It greatly improves the throughput in transforming a batch of operations. The comparative experimental results showed that POT outperforms the other well-known algorithms in a large collaborative workload. In future work, we try to extend our multi-core accelerated idea to other collaborative applications, such as CAD&Graphics systems [18–26].

Acknowledgment. This paper is supported by the National Science Foundation of China (Grant No. 61472289) and Hubei Province Science Foundation (Grant No. 2015CFB254).

References

1. Sun, C., Xia, S., Sun, D., Chen, D., Shen, H., Cai, W.: Transparent adaptation of single-user applications for multi-user real-time collaboration. ACM Trans. Comput. Hum. Interact. **13**(4), 531–582 (2006)
2. Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T., Englund, M.: Cored: browser-based collaborative real-time editor for java web applications. In: Proceedings of the ACM Conference on Computer Supported Cooperative Work, pp. 1307–1316. ACM (2012)
3. Nicolaescu, P., Derntl, M., Klamma, R.: Browser-based collaborative modeling in near real-time. In: The 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing, pp. 335–344 (2013)
4. Liu, H., He, F., Zhu, F., Zhu, Q.: Consistency maintenance in collaborative cad systems. Chin. J. Electron. **22**(1), 15–20 (2013)
5. Shao, B., Li, D., Gu, N.: A sequence transformation algorithm for supporting cooperative work on mobile devices. In: Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, pp. 159–168. ACM (2010)
6. Ressel, M., Nitsche-Ruhland, D., Gunzenhäuser, R.: 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, pp. 288–297. ACM (1996)
7. Randolph, A., Boucheneb, H., Imine, A., Quintero, A.: On synthesizing a consistent operational transformation approach. IEEE Trans. Comput. **64**(4), 1074–1089 (2015)
8. Li, D., Li, R.: An admissibility-based operational transformation framework for collaborative editing systems. Comput. Support. Coop. Work **19**(1), 1–43 (2010)
9. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: ACM SIGMOD Record, vol. 18, pp. 399–407. ACM (1989)
10. Suleiman, M., Cart, M., Ferrié, J.: Serialization of concurrent operations in a distributed collaborative environment. In: Proceedings of the ACM Conference on Supporting Group Work: The Integration Challenge, pp. 435–445. ACM (1997)
11. Oster, G., Urso, P., Molli, P., Imine, A.: Data consistency for p2p collaborative editing. In: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work, pp. 259–268. ACM (2006)

12. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978)
13. Imine, A.: Flexible concurrency control for real-time collaborative editors. In: The 28th International Conference on Distributed Computing Systems Workshops, pp. 423–428. IEEE (2008)
14. Michael, M.M.: Hazard pointers: safe memory reclamation for lock-free objects. IEEE Trans. Parallel Distrib. Syst. **15**(6), 491–504 (2004)
15. Ahmed-Nacer, M., Ignat, C.-L., Oster, G., Roh, H.-G., Urso, P.: Evaluating crdts for real-time document editing. In: Proceedings of the 11th ACM Symposium on Document Engineering, pp. 103–112. ACM (2011)
16. Li, D., Li, R.: An operational transformation algorithm and performance evaluation. Comput. Support. Coop. Work **17**(5–6), 469–508 (2008)
17. Shao, B., Li, D., Gu, N.: A fast operational transformation algorithm for mobile and asynchronous collaboration. IEEE Trans. Parallel Distrib. Syst. **21**(12), 1707–1720 (2010)
18. He, F., Han, S.: A method and tool for human-human interaction and instant collaboration in cscw-based cad. Comput. Ind. **57**(8), 740–751 (2006)
19. Jing, S.-X., He, F., Han, S.-H., Cai, X.-T., Liu, H.-J.: A method for topological entity correspondence in a replicated collaborative cad system. Comput. Ind. **60**(7), 467–475 (2009)
20. Huang, Z., He, F., Cai, X., Zou, Z., Liu, J., Liang, M., Chen, X.: Efficient random saliency map detection. Sci. China Inf. Sci. **54**(6), 1207–1217 (2011)
21. Liu, H., He, F., Cai, X., Chen, X., Chen, Z.: Performance-based control interfaces using mixture of factor analyzers. Vis. Comput. **27**(6), 595–603 (2011)
22. Li, X., He, F., Cai, X., Zhang, D.: Cad data exchange based on the recovery of feature modelling procedure. Int. J. Comput. Integr. Manuf. **25**(10), 874–887 (2012)
23. Li, X., He, F., Cai, X., Zhang, D., Chen, Y.: A method for topological entity matching in the integration of heterogeneous cad systems. Integr. Comput. Aided Eng. **20**(1), 15–30 (2013)
24. Cheng, Y., He, F., Cai, X., Zhang, D.: A group undo/redo method in 3d collaborative modeling systems with performance evaluation. J. Netw. Comput. Appl. **36**(6), 1512–1522 (2013)
25. Cai, X.T., He, F.Z., Li, W.D., Li, X.X., Wu, Y.Q.: Encryption based partial sharing of cad models. Integr. Comput. Aided Eng. **22**(3), 243–260 (2015)
26. Zhang, D.J., He, F.Z., Han, S.H., Li, X.X.: Quantitative optimization of interoperability during feature-based data exchange. Integr. Comput. Aided Eng., 1–20 (2015, preprint)