P2P-based Locking in Real-Time Collaborative Editing Systems

Ruixuan Li, Guangcan Yu, Zhengding Lu, Wei Song
College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P. R. China
Email: rxli@hust.edu.cn, xgygcan@tom.com, zdlu@hust.edu.cn, weisong@smail.hust.edu.cn

Abstract

Real-time collaborative editing systems allow a group of users to view and edit the same document at the same time from geographically dispersed sites connected by communication networks. Consistency maintenance in concurrent access of shared objects is one of the key issues in designing these types of systems. This paper proposes a P2P-based locking schema for concurrency control in internet-based real-time collaborative editing systems. This schema can distribute locking services to all collaborative sites and quickly find out the locking service nodes of given shared objects. The characteristic of distributed locking services solves the hot spot problem in centralized locking schema, and the capability of quickly locating locking service nodes resolves the broadcast communication problem in the existing distributed locking schemas. It also provides the basic locking function in collaborative editing systems and can be used in varied locking polices.

Keywords: *Peer-to-Peer (P2P), Locking, Collaborative Editing.*

1. Introduction

Real-time collaborative editing systems are groupware systems that allow a group of users to view and edit the same text/graphics/image/multimedia document at the same time from geographically dispersed sites [1, 2]. To meet the requirements of high responsiveness in the Internet environment, it seems that the only way is to adopt a replicated architecture for the storage of shared documents. Shared documents are replicated at the local storage of each collaborating site, so editing operations can be performed at local sites immediately and then propagated to remote sites [3]. Concurrency control techniques are required to ensure that the state of the shared document in the replication architecture remains consistent even when users attempt to modify the document simultaneously in a group editing environment. To support concurrent editing in the replication architecture, several inconsistency problems have to be dealt with [2], such as divergence, causality violation and intention violation.

Generally, there are two kinds of concurrency control techniques to support concurrent editing in the replication architecture. The first technique allows conflict to occur and provides a conflict resolution mechanism to accommodate the effect of all operations in a consistent way. This kink of technique includes multiversioning [3, 4] and serialization [5]. These techniques support multiple users editing the same object at the same time, but can not control the amount of users. If many users happen to edit the same object at the same time, it will become very difficult to resolute conflicts. The techniques claim to be high responsiveness, but relative long network delay may cause an anomaly at the user interface. The second technique uses a conflict prevention approach based on locking. When a user edits an object, he places a lock on the object, which will prevent other users from generating conflicting operations on the same object. Though the locking technique is also afflicted with network delay, locking does have the merit of preventing conflicts from happening. Locking is actually complementary with optimistic concurrency control strategies such as multiversioning.

It order to make locking work, methods must be used to keep track of which objects have been locked so that permissions for locking requests can be granted or denied accordingly. Generally, locking approaches can be classified into two categories: centralized and distributed. Ensemble [6], GroupKit [7] and Suite [8] are collaborative editing system using centralized locking. Centralized locking may easily lead to the occurrence of "hot spots". Hot spots may occur at any time when a large number of clients want to access a single server simultaneously [9]. If the locking server is not able to deal with the requests of all these clients simultaneously, the system responsiveness may significantly degrade. Reduce [10], Grace [11] and CDL [12] are collaborative editing systems using distributed locking approach. In these systems a global lock table is maintained in all collaborative sites. Locking operations on objects are broadcasted to all sites. The message broadcast mechanism obvious increase network load, and maintain the global lock table in all collaborative sites is difficult. In Reduce and

1-4244-0963-2/07/$25.00 ©2007 IEEE.

Grace, locking conflicts are resolved by coordinator-based protocol by means of a centralized coordinator and a locking transformation algorithm. This will also makes the occurrences of hot spot problem. A token is usually used in distributed locking scheme [13]. When a site acquires a lock, it gets a lock-specific token. Only one site can have the token at a time. The coarse granularity of locking makes collaborative system poor responsiveness.

To solve these problems, we propose a novel P2P-based locking schema. Every collaborative site is a client. It sends locking request to the other site when starting to edit an object. Every collaborative site is also a server that responses for locking requests of other sites and provides locking services. Essentially, we use a consistent hash function to partition all objects of the shared document [9]. Every collaborative site is responsible for locking services of some partitions of objects. For every collaborative site providing locking service for others sites, hot spot problem is avoided and the fine granularity locking is available. In fact, every object of a shared document can possess a lock to improve concurrency and responsiveness of collaborative system.

In the paper, Chord is extended to implement peer-to-peer (P2P) based locking. Chord is a scalable peer-to-peer lookup protocol for Internet applications [14]. It provides fast distributed computation of a hash function, mapping keys to nodes responsible for them. Chord assigns keys to nodes with consistent hashing, which has several desirable properties. With high probability, the hash function balances load (all nodes receive roughly the same number of keys). Also with high probability, when the nth node joins (or leaves) the network, only an $O(1/n)$ fraction of the keys are moved to a different location, which is nearly the minimum cost to maintain a balanced load. In a word, Chord provides support for just one operation: given a key, it maps the key onto a node. These characteristics make it suitable and easily apply to P2P-based locking in real-time collaborative editing systems.

The rest of this paper is organized as fellows: Section 2 describes P2P-based locking schema using the form of pseudo code. Section 3 discusses some special problems in the locking schema. Finally, Section 4 concludes the paper.

2. P2P-based locking schema

2.1. Basic concepts

A real-time collaborative editing system can be abstractly represented as follows: users (U) residing on different nodes (N) work on a shared document and operate objects (O) of the shared document. The basic idea of the locking approach is as follows: every object

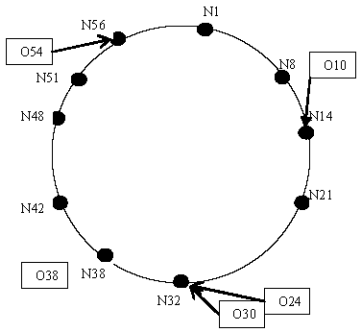


Figure 1. Chord circle consisting of ten nodes and five objects

of a shared document possesses a corresponding lock. If users operate on the object, the corresponding lock must be acquired. Every node provides locking services for some objects, and the load of nodes is approximately balance.

Every object has a unique identifier, and every node has a unique IP address. Through using the consistent hash function, each identifier and IP are transferred to an m-bit identifier. We will use the term "object" to refer to both the original object and its identifier under the hash function since the meaning of the object will be cleared from context. Similarly, the term "node" will refer to both the node and its identifier under the hash function. Chord provides support for mapping the given object onto a node, and the mapping node provides locking service for the given object. Consistent hashing assigns objects to nodes as follows. Identifiers of nodes are ordered on an identifier circle (Chord circle) modulo 2^m. Object o is assigned to the first node whose identifier is equal to or follows (the identifier of) o in the identifier space. Figure 1 shows a Chord ring with m=6 (modified from Chord). The Chord ring has ten nodes and provides locking services for five objects. Node 14 for object 10, similarly, node 32 for objects 24 and 30, node 38 for object 38, and node 56 for object 54.

The primary characteristic of P2P network is that node joins or leaves irregularly. This makes it dynamic for the nodes to provide locking services for given objects. It's the major problem that we try to solve in P2P-based locking schema.

2.2. P2P-based locking algorithms

The complete locking operations include those of client and server sides. We will describe the locking algorithms on server and client view respectively as follows.

2.2.1. Locking algorithms of server side

In Chord circle, each node knows its successor and predecessor node. We add a lock table to each node, which indicates those objects the node provides locking services for. If a node provides locking service for an

object, we say the node is the services node of the object in the following description. The object whose locking service is provided by a node will be called service object of the node. The lock table is expressed as *olock* (*oid*, *islocked*), the *oid* represents the identifier of objects, *islocked* represents lock status of objects. The lock table of each node has a backup on successor node of each node. The backup lock table is expressed as *olock_b*(*oid*, *islocked*).

Figure 2(a) shows the pseudo code of the server side lock algorithm. Remote calls and variable references are preceded by the remote node identifier, while local variable references and procedure calls omit the local node. In Figure 2(a), *n.foo*(.) denotes a remote procedure call (RPC) of procedure *foo* on node *n*, while *n.bar*, without parentheses, is an RPC to fetch a variable *bar* from node. *olock* represents the lock table of the current node and *successor.olock* represents the lock table of successor node. For convenience, we define some operations on the lock table. The operation *olock.getIslocked*(*oid*) get *islocked* property of the object *oid*. If the object *oid* is not in *olock*, we get false. The operation *olock.modify*(*oid, true*) modify *islocked* property of the object *oid*. If the object *oid* is not in *olock*, the object is appended to *olock* and the property will be modified. The operation *olock.delete*(*oid*) remove the object *oid* from *olock*. The operation *olock.dump*(*l*) copy all entries in *olock* to another lock table *l*. The operation *olock.get*(*oid*) gets the entry in *olock* of the object *oid*. The operation *olock.insert*(*e*) inserts an entry *e* into *olock*.

The operation *s_lock* represents how node *n* provides locking services for object *oid*. If object *oid* has already been locked, the operation will fail. Otherwise, modify its property (if it isn't in *olock* then first added to), set *islock* properties true. At the same time, it needs to modify the property of the object *oid* on the successor node of node *n*, set *islock* property true. So, when node *n* leaves abruptly, the successor of *n* can immediately take its work.

Figure 2(b) shows how object *oid* is unlocked on node *n*. Similar with the lock operation, set *islock* property of object *oid* false both on node *n* and its successor.

If an object is deleted in collaborative editing system, its corresponding entry in *olock* of the service node must be removed. Figure 2(c) shows the removal operation.

The primary characteristic of P2P network is that node joins or leaves anomalistically. If a node joins (or leaves) Chord circle, the service node of some objects will change from one node to another and corresponding entries in *olock* must also be moved. Although locking service nodes of some objects are changed, it should be transparent to the client nodes. Chord interacts with the upper application in two main ways. First, Chord library provides a *lookup*(*key*)

```
n.s_lock (oid)
 1: status=olock.getIslocked(oid);
 2: if status==true return false;
 3: olock.modify(oid,true);
 4: successor.olock_b.modify(oid,true);
 5: return true.
```

(a) lock operation

```
n.s_unlock (oid)
 1: status=olock.getIslocked(oid);
 2: if status==fase return false;
 3: olock.modify(oid,false);
 4: successor.olock_b.modify(oid,false);
 5: return true.
```

(b) unlock operation

```
n.s_dellock (oid)
 1: olock.delete(oid);
 2: successor.olock_bak.delete(oid).
```

(c) delete object

```
n.append(oidS)
 1: olock_b.dump(olock);
 2: for each oid ∈ oidS
 3:     successor.olock_b.insert(olock.get(oid));
 4: olock_b.clear();
 5: predecessor.olock.dump(olock_b).
```

(d) node leave

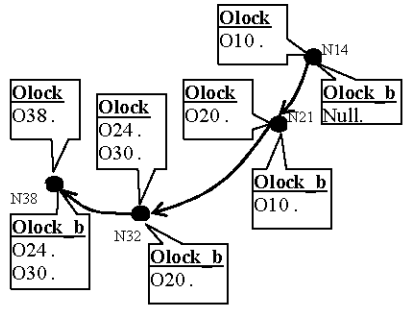
```
n.remove(oidS)
 1: olock_b.dump(predecessor.olock_b);
 2: olock_b.clear();
 3: for each oid∈ oidS
 4:     predecessor.olock.insert(olock.get(oid));
 5:     olock_b.insert(olock.get(oid));
 6:     olock.delete(oid);
 7:     successor.olock_b.delete(oid).
```

(e) node join

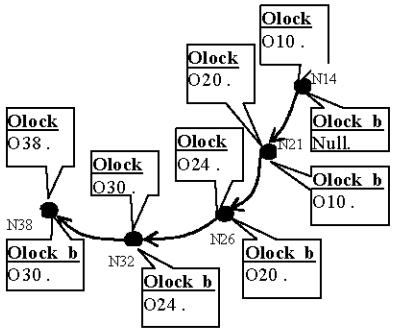
Figure 2. Locking algorithms of server side

function that yields the IP address of the node responsible for the key. Second, the Chord software on each node notifies the application of changes in the set of keys that the node is responsible for. We can use *lookup*(*oid*) to find out the locking service node of the object *oid*, and move related entries of *olock* to correct node when Chord notifies changing.

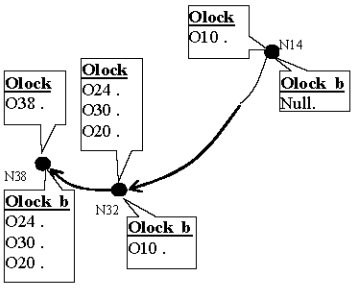
When the predecessor of node *n* leaves abruptly, node *n* becomes the service node of objects whose locking services were provided by the predecessor before. Figure 2(d) shows the operations when some



(a) initial state



(b) node 26 joins



(c) node 21 leaves

Figure 3. Changing of *olock* and *olock_b* on related nodes while node joins and leaves

objects append to node *n*. The *olock_b* on node *n* contains locking status of service objects on the predecessor node. When the predecessor leaves, all entries in *olock_b* are inserted into *olock*. If node *n* leaves abruptly, to avoid abandoning locking status of objects, the successor of node *n* backup *olock* of node *n* to its local *lock_b*. Similarly, backup the *olock* on the new predecessor node to *olock_b* of node *n*.

When a node joins Chord circle and becomes predecessor of node *n*, some service objects of node *n* must be moved to the newly joining predecessor. Figure 2(e) shows the operations when some objects are removed from node *n*. *oidS* is a set of objects will be removed from node *n*. This set can be obtained from Chord. If *n1* is the predecessor node of *n* before the new node joins, *olock_b* of node *n* is the backup of node *n1*. After a new node *n2* interpose between node *n* and *n1*, node *n2* become the successor of node *n1* and the predecessor of node *n*. Hence, all entries of *olock_b* are copied to the predecessor *n2* first by function

```
c_lock(oid)
 1: n=lookup(oid);
 2: isOK=n.s_lock(oid);
 3: return isOK.
```

(a) lock operation

```
c_unlock(oid)
 1: n=lookup(oid);
 2: isOK=n.s_unlock(oid);
 3: return isOK.
```

(b) unlock operation

```
c_dellock(oid)
 1: n=lookup(oid);
 2: isOK=n.s_dellock(oid);
 3: return isOK.
```

(c) delete object

Figure 4. Locking algorithms of client side

dump(.). Objects in set *oidS* become service objects on the predecessor *n2*. Entries of *olock* whose *oid* property is element of set *oidS* will be inserted into *olock_b* and *olock* of the predecessor *n2*. For those objects in set *oidS* which are not any longer service objects of node *n*, they will be removed from *olock* of node *n* and *olock_b* of the successor node.

Figure 3 gives an example which describes changing of *olock* and *olock_b* on related nodes when some nodes join or leave the Chord circle.

2.2.2. Locking algorithms of client side

When a user wants to edit an object, he must lock the object at appropriate time. Figure 4(a) illustrates how to lock an object. The function *lookup*(.) provided by Chord library can determine the service node *n* of an object *oid*. The operation *s_lock* is then called on node *n* to lock the object *oid*, and result of locking will be returned to applications.

After finishing editing an object, a user must also unlock the object at appropriate time. Figure 4(b) shows the unlock process of the object *oid*. Similar with the *c_lock* operation, first invoke function *lookup*(.) to get the service node *n* of the object *oid*, and then operation *s_unlock* is called on node *n* to unlock the object *oid*.

If an object is deleted from the shared document, the lock state of the object must be cleared. Figure 4(c) illustrates how to clear lock state of a deleted object on the service node.

3. Discussions

The fundamental idea of the proposed P2P-based locking mechanism is to distribute locking services to all collaborative sites and quickly find out the service

node of an object. From different points of view, locks can be classified into four categories: compulsory lock, optional lock, explicit lock and implicit lock. In broad sense, locks can also be classified into two categories: pessimistic and optimistic locks. P2P-based locking schema provides the basic locking functions in collaborative editing systems. In theory, it can be used in all above locking schemas.

The major problem of P2P-based locking is that a node n and its successor leave simultaneously. This will lead to service objects of node n loss its lock states. We give the following locking policy to solve the problem.

Before a user begins editing an object, the object is locked. After the lock operation is successful, the user can operate the object one or more times. When finishing editing, the object is unlocked. Since P2P-based locking method provides support for an object-level fine granularity lock, there may be many objects in a shared document. Though users lock objects relative long time, the interoperability of collaborative editing systems will be less decreased.

If a user has already locked an object and is operating on the object, simultaneously leaving of the service node of the object and its successor will lead to the following situation. The user continues to operate on the object without knowing the object automatically unlocked on the service node. If no other users want to edit the object until the user issues the unlock operation on the object, all operations are successfully and no client sites are aware of the loss of service node.

Figure 5(a) gives an example of the scenario. At time T_1, an object o is locked by site s_1. Then operation op_1 is forced on the object on s_1 and spread to the shared document on site s_2 and s_3. At time T_2, the service node of the object o and its successor leave simultaneously, and no node keeps the lock status of the object. However, site s_1 does not know that and continues carrying operation op_2 on the object and op_2 will be also spread to site s_2 and s_3. At time T_3, site s_1 unlock the object, though the unlock operation is meaningless. Since no other sites lock the object and operate on it between time T_2 and T_3, the operation op_1 and op_2 are both successful.

Figure 5(b) illustrates another scenario that a user wants to edit the object before the object is unlocked. If site s_3 locks the object at time T between time T_2 and T_3, the lock operation will be successful because the service node of the object o and its successor left simultaneously at time T_2. If site s_3 receives operation op_2 on the object o after lock it, it is obviously illogic. At time T', site s_3 relocks the object to confirm it has already locked the object. Then site s_3 undoes the operation op_2 and spreads it to shared document on s_1 and s_2. When site s_1 received the undo operation, it knows the service node has left and no longer operates on the object. Site s_3 then can normally operate on the object o, carrying operation op_3 on the object o and

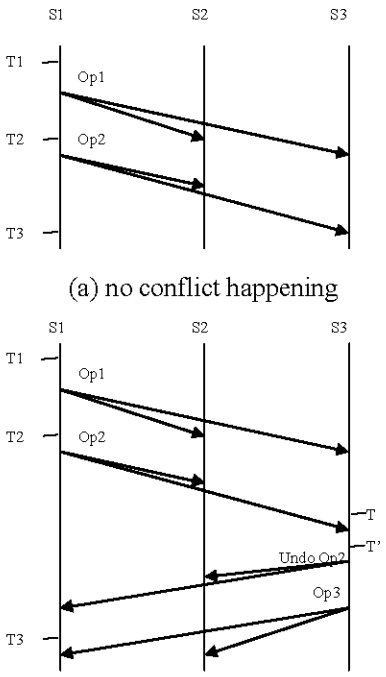


Figure 5. Scenarios of two nearby nodes leaving simultaneously

spreading it to site s_1 and s_2.

In some real-time collaborative editing system, users may be tagged with different priorities. Users with higher priority may want to extort control of objects from other users with lower priority. Our P2P-based locking can be applied to locking with priority. Add a *priority* field to object lock table, and then *olock* can be expressed as *olock* (*oid, islocked, priority*). When a user wants to lock an object, his priority is provided. If the priority is higher than the priority of the object in the *olock*, then the user locks the object and gains control of the object. Otherwise, the lock operation will fail.

4. Conclusion

In this paper, a P2P-based locking scheme is proposed for concurrency control in Internet-based real-time collaborative editing systems. In the locking scheme, every collaborative site acts as a client and sends locking requests to the service sites before starting to edit an object. Every collaborative site also acts as a server, responses for locking requests of service objects and provides locking services. Our P2P-based locking mechanism provides support of distributing locking services to all collaborative sites and quickly finding out the service node of an object. The distributing characteristic resolves hot spot problem in centralized locking scheme, and the function of quickly locating service node resolves broadcast communication problem in existing distributed locking approaches. Excellent characteristic of Chord can

promise balance load on all collaborative sites with high probability and small data need to be moved when new sites join or leave. In addition, the P2P-based locking scheme can be used in varied locking polices, such as compulsory locking, optional locking, explicit locking and implicit locking, since it provides the basic locking functions in the collaborative editing systems.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grant 60403027, Natural Science Foundation of Hubei Province under Grant 2005ABA258, Open Foundation of State Key Laboratory of Software Engineering under Grant SKLSE05-07.

References

[1] C. Ellis, S. Gibbs and G. Rein, "Groupware: Some issues and experiences", *CACM'34*, Jan 1991, pp.39–58.

[2] C. Sun and C. Ellis, "Operational transformation in real-time group editors: Issues, algorithms, and achievements", *In Proceedings of the ACM Conference on Computer Supported Cooperative Work*, ACM, New York, 1998, pp.59-68.

[3] C. Sun and D. Chen, "Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems", *ACM Transactions on Computer-Human Interaction*, 2002, (9)1: 1-41.

[4] D. Chen and C. Sun, "A distributed algorithm for graphic object replication in real time group editors", *In Proceedings of the ACM Conference on Supporting Group Work*, ACM, New York, 1999, pp. 121-130.

[5] R. Kanawati, "LICRA: A replicated-data management algorithm for distributed synchronous groupware application". Parallel Compute'22, 1997, pp.1733-1746.

[6] R.E. Newman-Wolfe, M.L. Webb and M. Montes, "Implicit Locking in the Ensemble Concurrent Object-Oriented Graphics Editor", *Proceedings of the ACM conference on Computer supported cooperative work*, ACM, Ontario, Canada, Nov 1992, pp.265-272.

[7] S. Greenberg and D. Marwood, "Real Time Groupware as a Distributed System: Concurrency Control and Its Effect on the Interface", *Proceedings of the ACM CSCW Conference on Computer Supported Cooperative Work*, ACM, North Carolina, Oct 1994, pp.207-217.

[8] J. Munson and P. Dewan, "A Concurrency Control Framework for Collaborative Systems", *In Proc. of ACM Conference on Computer Supported Cooperative Work*, Nov. 1996, pp.278-287.

[9] D. Karger, E. Lehman and T. Leighton, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web", *In Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, May 1997, pp. 654-663.

[10] C. Sun, "Optional and Responsive Fine-Grain Locking in Internet-Based Collaborative Systems", *IEEE Transactions on Parallel and Distributed Systems*, Sep. 2002, pp. 994-1008.

[11] D. Chen and C. Sun, "Optional and Responsive Locking in Distributed Collaborative Object Graphics Editing Systems", *Proceedings of the First International Conference on Web Information Systems Engineering (WISE'00)*, 2000, pp. 414-418.

[12] X. Xu, J. Bu and C. Chen, "Distributed dynamic-locking in real-time collaborative editing systems", *Lecture notes in computer science*, Springer , 2004, pp.271-279.

[13] A. Prakash and H. Shim, "DistView: Support for building efficient collaborative applications using replicated objects", *In Proceedings of the Fifth Conference on Computer Supported Cooperative Work*, ACM, Toronto, Canada, Oct. 1994, pp.153-164.

[14] I. Stoica, R. Morris and D. Liben-Nowell, "Chord: A scalable peer-to-peer lookup protocol for internet applications", *In Proceedings of the ACM SIGCOMM*, California, August 2001, pp.149-160.