Distributed Dynamic-Locking in Real-Time Collaborative Editing Systems*

Xianghua Xu, Jiajun Bu, Chun Chen, and Yong Li

College of Computer Science, Zhejiang University
Hangzhou 310027, P.R.China
{xuxh_cs,bjj,cchen,lyzju}@cs.zju.edu.cn

Abstract. In this paper, a Customizable and Dynamic Locking (CDL) scheme is proposed for concurrency control in Internet-based real-time collaborative editors. The idea of dynamic-locking is that: locking mechanism dynamically determines locking set according to locking policies and latest collaborative activities happened in the shared workspace, and pre-locks objects for succeeding editing and preventing from other user's edit. Dynamic locking is optional: user decides whether and when to use locking mechanism. In the proposed scheme, locking policy is separated from locking mechanism. Locking policies can be customized for different collaboration tasks. Locking scope is dynamically determined according to locking policies and collaborative activities among users. Multiple users can select different policies in the same collaborative session, and change locking policies at different phases of collaboration as well. Protocols and algorithms for locking conflict resolution and consistency maintenance are also presented in this paper.

1 Introduction

Internet-based collaborative editing systems are a special class of distributed applications which allow a distributed community of users to simultaneously edit a shared document[9]. A replicated architecture is usually adopted in which the shared documents are replicated at local storage of each collaborating site. With the replicated architecture, multiple users can concurrently edit their local copies of the shared document and get their operations reflected on their local interfaces immediately. One of the most significant challenges in the design and implementation of replicated collaborative editing systems is consistency maintenance of replicated documents in face of concurrent updates[10].

Locking is a widely used concurrency control technique in distributed computing and database systems to ensure data integrity by prohibiting concurrent conflicting updates on shared data objects [8]. Locking has also been used for consistency maintenance of shared documents in various collaborative editing systems. From users' points of view, locking can be classified into two categories: compulsory locking and

* This material is based upon work funded by Zhejiang Provincial Natural Science Foundation of China under Grant No. Z603231.

G.-J. de Vreede et al. (Eds.): CRIWG 2004, LNCS 3198, pp. 271–279, 2004.

Table 1. Compulsory-optional vs explicit – implicit.

	Explicit	**Implicit**
Compulsory	√	√
Optional	√	

optional locking. Locking is compulsory in the sense that a lock must be requested before editing an object, whereas a system with optional locking allows users to update any unlocked objects without necessarily requesting a lock on it. Locking can also be classified into explicit lock and implicit lock. In explicit locking, lock and unlock are requested by a user before and after an editing operation, whereas implicit locking is requested by systems when an object is selected and released after deselecting. The relationship between compulsory-optional and explicit-implicit locking is shown in Table 1. The existing compulsory locking can be explicit or implicit either, whereas optional locking can only be explicit. In previous researches, the optional-implicit locking was unmentioned.

In explicit locking scheme, the granularity can be object, region, or whole document. It's determined by users at runtime. In implicit locking scheme [7], the granularity is statically determined by systems in advance. Obviously, explicit locking is more suitable for collaborative environment as it is more flexible than implicit locking. However, explicit locking leaves too much burdens on user since users need to request for lock before updating. Comparatively, implicit locking mechanism releases users from locking and unlocking, while losing flexibility and adaptability.

In optional locking, when to lock and how to lock are all decided by user at runtime. Because multiple users are often editing different regions in many occasions, these will not cause concurrent editing conflict problems. Therefore, optional lock is more appropriate choice for collaborative application than compulsory locking. However, this requires users to entirely determine when to lock and how to lock. Moreover, multi-users' working areas are often interweaving with each other and constantly changing in the evolution of collaboration. Hence, how to determine the locking scope is a burden on user.

Therefore, we propose a dynamic locking approach in this paper. The main idea is: users can define the locking policy to be used in the system and decide when to use dynamic locking; lock mechanism dynamically determined locking scope at runtime according to the locking policies user defined. The key issues in dynamic locking approach is how to dynamically determine the locking scope and how to resolve the locking conflict and maintain the consistency of locking data in a distributed collaborative environment. In this paper, we focus attention on the locking conflict problem. How to determine the appropriate locking scope will be discussed in another paper.

2 Dynamic-Locking Approach

2.1 Basic Definitions

A real-time collaborative editing system can be represented as (*S*, *U*), where *S* is the *shared workspace*, *U* is a set of participants. There is a collection of *objects* (*O*) exist-

ing in *S* shared by *U*. *S* is a metaphor for compositing together with *U* to manipulate a collection of objects (*O*) and their attributes. The concepts related to user's activities in *S* are defined as follow:

Operation (*Op*) is the action issued by user to manipulate the objects. Each operation is stamped by time-vector as defined in [3, 5]. A dependent (causal) relationship or independent (concurrent) relationship between any two operations can be determined according to operation's time-vector [9]. Further more, a real-time collaborative editing system is said to be consistent if it always maintains the three properties: convergence, causality-preservation and intention-preservation [10].

Location (*Loc*) is an object that the user currently selects or edits.

Lock set (*Ls*) is a set of objects locked by a user. For each user, a current locking set is maintained at each site, which represents the objects currently locked by that user.

Locking point (Lp) is defined as a locked object that is currently edited by lock owner. A user's Lp is a member of his locking set and also is user's Loc.

Context value (Con) represents U's relevancy degree on the local context affected by Op. *Con* value is used in calculation of locking set and conflict resolution protocol.

Lock table (*Lt*) is maintained at each site. There is an entry in lock table for each user, which points to the user's lock set.

2.2 The Architecture of Dynamic Locking Model

Locking mechanism is the core component in dynamic locking. When a user needs to utilize the locking mechanism, it will be enabled, and the specific locking policies are selected. The locking procedure is described briefly as follow:

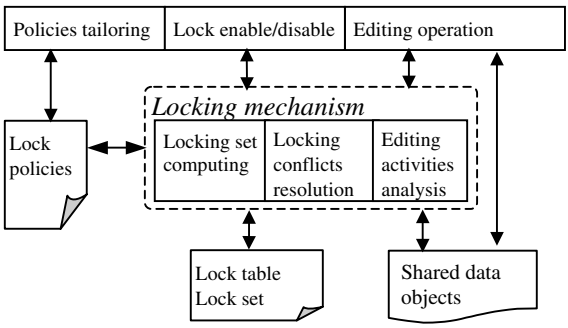

Fig. 1. Architecture of dynamic locking.

When a user selects, edits or creates an object, the *locking mechanism* calculates the user's locking set according to locking policies and user's latest editing activities, then, locks objects belong to the locking set, and then, time-stamp and broadcast the locking operation to other sites. When the user begins to edit next object, new locking set will be recalculated and locked, and the previous locks will be released.

When a remote locking operation is received, the *locking mechanism* performs conflict-check with operations in HB. If conflict occurs, it will be resolved according to the conflict resolution protocol. Otherwise, objects belonging to the locking set will be locked by the remote user.

For each editing operation, the locking mechanism does some works of activity statistics. The statistic result is used by itself for the calculation of locking set.

2.3 Example of User's Activity Analysis

When multiple users participate in an editing or design task (except the task like brainstorm), each user is often responsible for a subtask assigned. Although the subtask may be only a rough division of the collaboration task, the history of editing operation will present out the pattern of task division, and, such pattern can be represented as rough and irregular division of the shared workspace. Hence, we can extract the pattern by the analysis of users' activities. Here we propose a simplified method for a 2D shared workspace.

First, we divide the 2D shared workspace into M x N small rectangle grids, denoted as S[M][N]. For each collaborative session, we maintain an array, defined as $AF^U[M][N]$, for each user U. $AF^U[M][N]$ is used to express the frequency of U's editing activity affect on S[M][N] . Hence, $AF^U[M][N]$ represents a subtask pattern of user U mapped on S[M][N]. $AF^U[m][n]$ expresses the frequency of U's editing activities at sub-workspace S[m][n]. For a group users: u1, u2 and u3, which participate in the same editing session, $AF^{U1}[M][N]$, $AF^{U2}[M][N]$ and $AF^{U3}[M][N]$ represent the pattern of group editing work.

For a given user U and a Op issued by U:

$$AF^U[i][j] = AF^U[i][j] + 1, \text{ for all } S[i][j] \cap Op, \text{ where } I \in [0,M-1], j \in [0,N-1]$$

For an Op issued by user U, U's context relevance (Cr) for Op is defined as:

$$Cr^U = \sum_{\forall S[i][j] \cap Op} AF^U[i][j], i \in [0, M-1], j \in [0, N-1]$$

Given a group of users, u_1, u_2, … , u_k, participating in a group editing session. For an operation Op generated by user u_s, the *local context value* (*Con*) of Op is defined as:

$$Con_{U_s}^{Op} = \frac{Cr^{U_s}}{\sum_{t=1, t \neq s}^{K} Cr^{Ut}}, \quad Con_{U_s}^{Op} \in [0,1]$$

The context value *(Con)* represents U's relevance on the local context affected by Op. *Con* value is used in calculation of locking set and conflict resolution protocol in following section.

2.4 Locking Policy and Locking Set

Locking policy and *Context* value are two key factors in computation of locking set. Users can describe specific locking policy according to characteristics of application and subjective requirement. Locking policy is expressed as:

Policy(PolicyName, Type, [[scale] | [prediction]])
Type = [Rect | Circle | Irregular | Bool]
Scale = [RegionSize: 1,2,…,n]
Prediction = [Attr(Obj) = value] |[Prediction [Bool] Prediction]
[BoolPolicyName]
Bool = [and | or]

There are three types of locking policy: object's geometry constraints (rectangle, circle, irregular); object's attribute constraints (such as object owner, owner's privilege); combination of geometry and attribute constrains. In geometry constraints, a max region size and a scale are defined. The max region is dynamically mapped into actual region at runtime: (max-region, scale) X (*Con*-value) → (actual-region). Here we illustrate the locking policy with following examples:

- Policy(rule1, Rect, [[200,200]: 1, 2, 3]), define a rectangle scope which will be mapped to a actual scope by scale [1,2,3] and *Con*.
- Policy(rule2, Bool, [Owner(Op) = Owner(O)]), describe an attribute constraint: only the owner's objects included.
- Policy(combinedRule, Bool, [rule2 and rule3]), define a combined rule of rule1 and rule2.

Computation of locking set: when a policy was enabled, locking mechanism is responsible for determination of locking set. When a user is selecting or editing an object, a current locking set is calculated out simultaneously, then the objects belonging to the set are locked immediately and locking set is broadcast to other sites. However, the locking set may conflicts with a concurrent editing Op or locking set received from other sites in face of concurrent locking and editing operations.

3 Dynamic Locking Protocol and Conflict Resolution Protocol

3.1 Dynamic Locking Protocol and Algorithm

Dynamic Locking Protocol (DLP):

1. When a Locking set is generated at a local site:
   Locking set will be granted at local site and propagated to all remote sites. A locking subset, which belongs to old Locking set but not to new Locking set, is released.
2. When a Locking set arrives at a remote site:
   a. If it does not conflict with any Op in HB (History Buffer of executed operation), the new Locking set is granted and compared with old Locking set. A locking subset, which belongs to old Locking set but not to new Locking set, is released.
   b. If it conflicts with Ops in HB, the conflicts are resolved by conflict resolution protocol.

   According to DLP, we present a Dynamic Locking algorithm:

```
Function DynamicLock(Op, U)
/* Op: a local/remote operation currently executed. U:
Op's issuer; Ls(U): U's locking set, depicts a group of
object locked by U;  Loc(U): the location of U in S; HB:
history buffer of executed operation. */
//for the local generated operation
If (Op is a local op) Then
 //How to compute context value and locking set is
 //application-oriented problem, hence the algorithms
 //will not given here
 Con = ComputeContext(Op);
 NewLs = ComputeLs(Op);
 //a subset of objects needs to be locked and unlocked
 Ladd = NewLs – Ls(U);
 Lfree = Ls(U) – NewLs;
 Ladd->lock();
 Lfree->unlock();
 // Pack Ls and Con into Op which will be sent to remote
site
 Op->Ls = NewLs;
 Op->Con = Con; Broadcast(Op);
 //setting the new lock set of U
 Ls(U) = NewLs;
 HB->Add(Op);
Else //for the remote operation
 //If Op is a concurrent operation with other operation
 If (IsConcurrent(Op, HB)) Then
  Resolve conflict by LCRP protocol;
 Else
  NewLs = Op->Ls;
  //a subset of objects needs to be locked and unlocked
  Ladd = NewLs – Ls(U);
  Lfree = Ls(U) – NewLs;
  Ladd->lock();
  Lfree->unlock();
 EndIf
 HB->Add(Op)
EndIf
```

3.2 Locking Conflict Resolution Protocol and Algorithm

The concurrent conflicts are resolved according to *Locking Conflict Resolution Protocol (LCRP)*:

For each Op in HB conflicting with remote Op,

1. If Op's Lock point is same as remote op's Lock point, then conflict is resolved by negotiation.
2. If Con(Op) = Con(remoteOp) then the conflicting subset of Ls(U(Op)) / Ls(remoteOp) will be released/ungranted.
3. If Con(Op) > Con(remoteOp) then the conflicting subset of Ls(remoteOp) will be ungranted.
4. If Con(Op) < Con(remoteOp) then the conflicting subset of Ls(U(Op)) will be released.

Fig.2 illustrates conflicts may be occurred in dynamic locking, yellow and green areas represent the locking set of op1 and op2 respectively, blue and red dots represent the locking points of op1 and op2 respectively. Suppose Con(op1)=Con(op2). In fig.2a, both locking points are out of conflict part of locking set, the conflicts are resolved according to rule 1 of LCRP. In fig.2b, both locking points are the same object, it is resolved by negotiation (rule 2). In fig.2c, op2's locking point is within the conflict part, it's resolved according to rule 2, but u2's locking point is still included in u2's locking set. In fig.2d, both locking points are in the conflict part, it's resolved according to rule 2, both locking points should be still included in their locking set respectively. Conflicts between locking and editing operation could be resolved similarly.

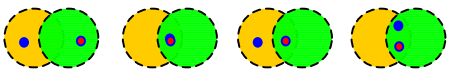

Fig. 2. A scenario of locking conflicts.

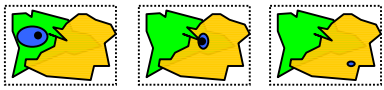

Fig. 3. A scenario of dynamic locking.

Fig.3 illustrates a simplified scenario of dynamic locking. Green and yellow are working areas of u1 and u2 respectively. Blue area represents u1's locking area and the size of lock set. When u1 edits objects in his working area (Fig.3a), his locking area will be max size defined in locking policy. When u1 edits objects in the intersected area of u1 and u2 (Fig.3b), his locking area will be adjusted to medium size. When u1 edits objects in the working area of u2 (Fig.3c), his locking area will be adjusted to minimum size.

4 Comparison to Related Work

A variety of locking schemes have been proposed to maintain consistency in collaborative editing systems. Ensemble[7], GroupKit[4], Suite[6] and REDUCE[8] are closely related to our work.

Ensemble adopts implicit-optimistic locking scheme, whereas GroupKit adopts explicit-optimistic locking scheme. They only support single locking granularity predefined in systems, hence these schemes cannot meet the requirement of collaboration because of lacking of flexibility.

Suite [6] provides multi-granularity locking on a general structured data model. It employs a flexible concurrent control model to associate data semantics. When conflict occurs, conflict resolution rules examine the operation rights to determine who gets the object. Specific lock policies use heuristics to decide whether a lock in use should be taken away and granted to another requester. Suite is an optimistic and compulsory locking scheme with multi-granularity. The granularity can be dynamically decided at runtime according to predefined policy and data semantics. However, Suite is designed for structured data model, hence it not suitable for semi-structured or unstructured data model adopted in many collaborative applications.

In Ensemble, GroupKit and Suite, locking is compulsory, in which users need to passively abide the locking constraints implemented in system and have to use locks before editing. In human-human collaborative interaction, however, when to lock and how to lock mainly depend on the collaborative users and the context they shared. Hence, the locking scheme should be user-centric, and it should be flexible so that user can decide when to lock and how to lock according to dynamic collaborative environment. Moreover, they all adopt a centralized server to resolve the locking conflicts.

The optional locking, presented in REDUCE [8], GRACE [2], is most related to our work. Optional locking is also a distributed, high responsive locking approach used in collaborative environment. However, the locking conflict is resolved by Coordinator-Based Protocol by means of a centralized coordinator and a locking transformation algorithm [8]. In our work, the locking conflicts are resolved in a fully replicated architecture, and no coordinator is needed. Moreover, in optional locking, the locking set is decided by users, but the locking set is dynamically determined by locking mechanism in our approach.

[1] presents an adaptive multi-granularity locking scheme used in object-oriented database. Multiple logical granularity units are chosen with the knowledge of the data model and changed dynamically during the execution of a transaction by means of escalation and de-escalation technique. It aims to improve the OOB' s throughput. However, it's not suitable for real-time collaborative system, because collaborative systems are more sensitive to response-time than to system throughput, and required to support unstructured activities with dynamic and context-specific consistency requirements.

5 Conclusions

In this paper, a Customizable and Dynamic Locking (CDL) scheme is proposed for concurrency control in Internet-based collaborative systems. CDL is fully distributed, highly responsive, dynamic and tailorable locking mechanism. In CDL scheme, locking policy are separated from locking mechanism. Locking policies can be customized to meet the requirements of users and application. Locking scope is dynamically determined according to locking policies and collaborative activities among users. Multiple users can select different policies in the same collaborative session. A user can change locking policies at the different phases in the same session. CDL is also optional: whether and when to use the locking mechanism is determined by users. When the CDL is enabled, it automatically decides the dynamic locking scope according to latest activities among collaborative users and locking policies selected. Protocols and algorithms are devised to resolve locking conflict and maintain consistency of global locking status.

The CDL scheme is implemented in CoDesign [11] to verify the feasibility. We are currently investigating the interface, activities analysis and usability issues related to CDL, including how the locking status is presented to the user, how to tailor the policy for different phases of collaborative works and users' subjective evaluation in

using CDL. The implementing issues, such as approach of activities analyzing, locking set calculation, conflicts resolution and consistency maintenance algorithm, etc, will be discussed in a follow paper.

References

1. C.T.K. Chang, *Adaptive Multi-Granularity Locking Protocol in Objectoriented Databases*, PhD Dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, 2002
2. 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)*. p.414-418.
3. C.A. Ellis and S.J. Gibbs. Concurrency Control in Groupware Systems. *In Proceedings of the ACM SIGMOD Conference on Management of Data*, May 1989. Seattle, WA, USA. p.399-407.
4. 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*, October 22-26. North Carolina: ACM Press.
5. L. Lamport, Time, Clocks, and the Ordering of Events in a Distributed System. *CACM*. 21,7 (1978): p. 558-565.
6. J. Munson and P. Dewan. A Concurrency Control Framework for Collaborative Systems. *In Proc. of ACM Conference on Computer Supported Cooperative Work*, Nov. 1996. p.278-287.
7. R.E. Newman-Wolfe, et al. Implicit Locking in the Ensemble Concurrent Object-Oriented Graphics Editor. *Proceedings of the ACM conference on Computer supported cooperative work*, November 1992. Toronto, Ontario, Canada. p.265-272.
8. C. Sun, Optional and Responsive Fine-Grain Locking in Internet-Based Collborative Systems. *IEEE Transactions on Parallel and Distributed Systems*. 13,9 (2002): p. 994-1008.
9. C. Sun and D. Chen, Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems. *ACM Transactions on Computer-Human Interaction*. 9,1 (2002): p. 1-41.
10. C. Sun, et al., Achieving Convergence, Causality-Preservation, and Intention-Preservation in Real-Time Cooperative Editing Systems. *ACM Transaction on Computer-Human Interaction*. 5,1 (1998): p. 63-108.
11. X. Wang, et al. Achieving Undo in Bitmap-Based Collaborative Graphics Editing Systems. *In Proceedings of 2002 ACM Conference on Computer Supported Cooperative Work (CSCW'02)*, November 16-20. New Orleans, Louisiana, USA.