Coshed/SAS: Groupware and User Interfaces

Jaeyoung A. Lee, Seok Hwan Yoon, Tonghyun Lee, Jieun Park

Abstract

Development in multimedia technology enhances Computer Supported Cooperative Work(CSCW) such as tele-conferencing and tele-education. These applications are also called groupware applications and usually composed of A/V conferencing module and shared working space such as group editors. Through this shared working space users can present, manipulate and save common artifacts interactively. Since users are allowed to operate on common artifacts simultaneously, conflicts between user transactions are bound to happen and can cause unnatural effects on user interfaces. In this paper we reviewed these problems and presented common solutions to concurrency control problems. We also presented Coshed/SAS, a group editor, as an example of groupware applications and related concurrency control problems.

1 Introduction

The human to computer interface has been seen from a purely personal perspective until recently. Applications generally provided single interface through which only one user interacted and interaction with other users was considered to be less important. Developments in networking and Computer Supported Cooperative Work (CSCW), however, has created situations where applications provide multiple interfaces simultaneously to a group of users. Users can co-work in a virtual shared workspace created and maintained by CSCW or cooperative applications. Cooperative applications have been considered as either collaboration transparent or collaboration aware[1].

Users of collaboration transparent applications are unaware that more than one interface is presented to them. In contrast, collaboration aware applications are devised to interact with a number of users by presenting a number of different user interfaces. Through the use of collaboration aware applications users should be able to work in an environment, virtual shared workspace, that simulates an ordinary collaboration environment in which users are highly informed of other users' activities. Imperfections of network, however, present serious obstacles in maintaining a unique copy of the workspace, which means users are presented with different realizations of the shared workspace. The user interface inconsistency is further distorted by simultaneous user inputs and can seriously degrade effectiveness of groupware applications as CSCW tools.

In this paper, we review the problem of concurrency control in groupware and its effects on user interfaces. The importance of concurrency control has been pointed out by many researchers[2,3,4,5,6,7]. Recently, the impacts of concurrency control on user interfaces are studied by some authors. We reviewed three basic concurrency control algorithms as the compositional basis for other derived algorithms. We also studied of the effects of the system topology and the replication scheme on the user interface. It is shown that all three factors- concurrency control algorithm, system topology and replication scheme-can seriously affect the logical consistency of the user interface and system performance.

2 Traditional concurrency control

Basic units of the client/server communication is *transaction*[7]. A transaction is the sequence of *atomic operations*, $\{a_1, a_2, ..., a_n\}$, where a_i is either read or write operation. Transactions can be committed or aborted by the server. Clients can transmit transactions independently from other clients' activities, which allows multiple transactions issued simultaneously. Concurrent transactions T_i and T_j are *in conflict* if atomic operations a_i of T_i and a_j of T_j access common data object and at least one of them is write operation. Confliting transactions can disturb data integrity. Various algorithms that detect and resovlve conflicts among concurrent transactions-concurrency control algorithms-are devised and applied with varying degree of success. Most of them are either combinations or modifications of three basic mechanisms: *locking*[8], *timestamp*[9], and *optimistic concurrency control*[10,11].

Locking A transaction can lock a set of objects for exclusive manipulation. As long as a data item is locked by a transaction, any other lock request is denied. If the lock is for reading the data item, the lock is *read lock* and *write lock* if it is for writing. If a transaction T_i has a read lock on a data item, a concurrent transaction T_j must not write until T_i commits or aborts.

timestamp ordering In timestamp ordering based concurrency control scheme, transactions are assigned with unique timestamps as they enter the server. A transaction with earlier timestamp has higher priority than transactions with later timestamp. Each operation in a transaction must be validated by the server when it is carried out according to the priority rule based on timestamp.

optimistic concurrency control Optimistic policy of concurrency control assumes that conflicting events rarely arrive and it is cost-effective to repair the adverse effects caused by the out-of-order transactions than suspend current operations and wait for the in-order transactions. Compared to locking, optimistic concurrency control can enhance concurrency of transaction processing but overhead of restoring occasional occurance of out-of-order transactions can also be substantial in some cases.

Distibuted transactions

Distributed transactions involve more than one server and data can be distributed over the servers or replicated. Besides concurrency control of transactions on local data items, there must be a cooperative procedure used by a set of servers that enables the servers to reach joint decision as to whether a transaction can be committed or aborted. As in the single server case, locking, timestamp ordering and optimistic concurrent control policy can be used.

3 Groupware and concurrency control

Groupware applications require another consideration over distriubed systems: Human-to-Machine interface. User action of ordinary distributed systems such as banking and file services is determined without knowledge of activities of other users of the system. In CSCW environment, however, users must be aware of presence of other participants before making any judgement. Transactions issued by users unaware of other users' activities can conflict with each other and adversely affect the virtual shared workspace, the depository of shared objects.

3.1 Real-time Groupware and user interaction

Interaction is an action that is influenced by the presence of, knowledge of, or the activities of another person. For routine collaborative activities, users can make decision with high knowledge of other participants. In CSCW environments, however, user knowledge is limited to the space(virtual shared workspace) provided by the system and user interaction based on limited information can be errant.

Consider a graphical editor shared by a group of users. User *A* and *B* simultanesouly operate on a centered circle by drawing a line passing through the center of the circle and moving the circle up, respcetively. After a relaxation time the system becomes a consistent state and the resulting digram is shown in the Figure 1. The final result mismatches both users' intentions and additional corrective actions must be taken by the users.

From this simple example it is obvious that the concept of unique shared workspace and unique shared documents is an idealization of real world situation. Network delay distorts the ideal emplementation of single copy shared workspace into possible multiple copy

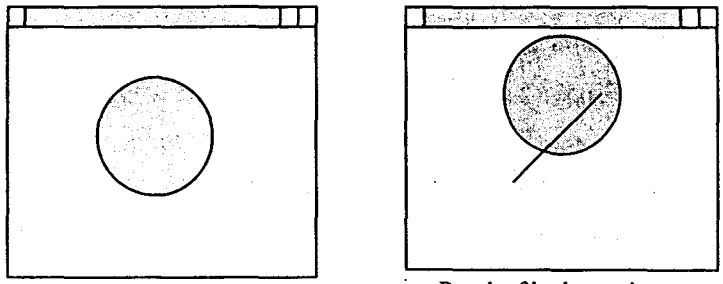

Figure 1: An example of a graphical shared editor

shared workspace. User action is displayed to other users with finite amont of time delay which can mislead users to make incorrect judgement of what others are doing.

Conflicts of user interactions can happen frequently and unexpectedly. Without proper mechanism of correcting conflicts, loss of logical context is unavoidable. Many algorighms have been devised to prevent or restore the system from conflicts, and applied to real applications with success of varying degree. The algorithms are combinations or variations of three basic algorithms given in Chapter 2. Each algorithm has unique impacts on user interfaces.

locking Locking is most common in groupware applications. Before a user operate an object he/she should request a lock on it. Depending upon the lock status of the shared object, the replication server responsible for the lock management verify the request and decide whether to grant or reject it. After a user acquires the lock, he/she has the exclusive right to manipulate it. Other users who want to modify the shared object have to wait until the lock is released by the owner. This behavior is radically different from collaboration transparent applications which allows users to work on any object at any time.

The efficiency of locking-based algorithms is determined by the granularity of locking. It is obvious that object table based lock is not ideal choice in most cases. Locking in

smaller units, on the other hand, does not always guarantee logical context. Consider the previous example of shared graphical editor where users *A* and *B* try to operate on the center circle. If each object is assumed to be locked independently the history of transaction processing is unaltered and users' requests are processed unhindered. The logical configurations both user intended to achive are still not achieved. On the other hand, if lock is on the object table, adding new object to the table and modifying an object conflict with each other and one must wait until the other is processed and shared workspace is updated.

timestamp ordering Transactions from clients in the timestamp ordering algorithms are processed in the order of the timestamp issued according to a pre-defined rule. Timestamp ordering achieves global serialization by issuing globally unique timestamps but the concurrent processing of user transactions is hindered completely.

optimistic concurrency control In groupware applications based on the optimistic concurrency control mechanism, a user can proceed with his/her operations on shared objects until the transaction is verified and committed. Transaction verification and commitment is handled by the replication servers. If all servers decide to commit and consensus among them is established, the originating client is notified and local object table is updated. The user proceeds with his/her work unware of the process. Therefore, as long as conflicting transactions rarely occur, the optimistic concurrency control algorithms give cooperation aware applications the closest feel of cooperation transparent applications. If consensus is not established, however, and the transaction is aborted, the client must process the abort message and roll back to the original consistent state. This process of rolling back can be quite hard to implement and confusing to users.

Table shown below summarizes this section. If user interaction is frequent and occurence of non-conflicting concurrent transactions such as read operations is common, optimistic concurrency control can guarantee the best performance. But if transactions are mostly write operations and conflicts among them occur frequently, locking algorithms must be used. In real implementation where transactions are combinations of read and write operations, appropriate mean between locking and optimistic concurrency control should be considered.

	Concurrency depends on	Effects on user interface	Degree of concurrency
locking	degree of granularity	jerky, slow response	low
timestamp ordering	implementation	slowest response	lowest
optimistic concurrency control	implementation	recovery operation can be unnatural	highest

3.2 Server or not

Benefits of server-based or centralized architectures have been praised by many authors. Advocates[12, 13, 14] point to such factors as simple implementation of concurrency control and reduction in necessary communication needed for a transaction to successfully update all user interfaces. Replicated architectures also have followers for the issue of the network latency and the fact that concurrenc processing of user inputs is drastically hindered by the introduction of a server. It is obvious that write operation intensive applications can benefit from the server-based architecture, andd read operation intensive applications can benefit from the replicated architecture.

However, strict distinction in terms of centralized and replicated architectures is naive. Distributed system architectures should be categorized according to two criteria, the geometry of the system and the repliation scheme. Employment of servers decides geometry of distributed system architectures. We refer servers as application instances independent from clients that manage front ends such as user interfaces. Servers can be located on client machines or on independent machines. In replicated systems, on the other hand, multiple copies of object tables are distributed over the network. A subset of $n+m$ machines of the system may maintain replicated copies of the object table, where n corresponds to the number of clients and m the number of servers. The geometry and the replication scheme are two different factors that must be regarded independent from each other. With this respect, above mentioned centralized system must be re-defined as single server systems with no replication of object tables and the replicated systems as serverless replicated systems. We can create many systems by varying the number of servers and the replication scheme. It is obvious for the reasons presented before that efficient systems should have replicated server architectures such as Fully Replicated Single Server

Architecture(FRSSA) employed by Coshed/SAS, a graphical shared editror developed by ETRI.

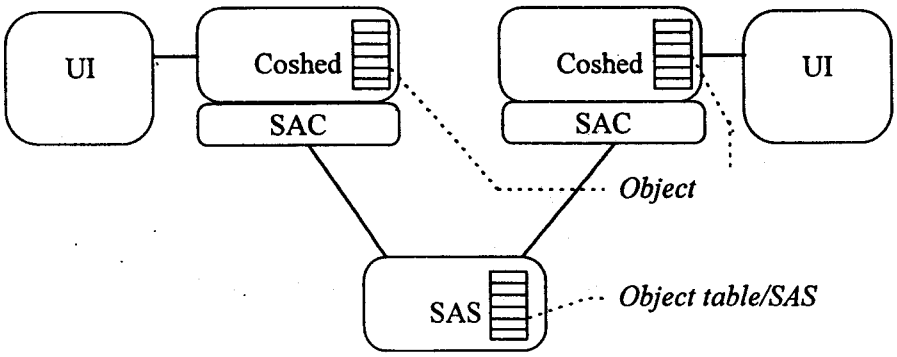

Figure 2: The structure of Coshed/SAS

3.3 Coshed/SAS and FRSSA

We have developed a shared graphical editor named Coshed/SAS. It allows more than one user to manipulate a common document and independently save it to a local mass storage device. In Figure 2, the SAC/SAS architecture of Coshed/SAS is schematically presented. User interaction is viewed as if it is happening in a virtual shared workspace called Shared Area, hence the name SAS(Shared Area Server). Since the Coshed/SAS system maintains fully replicated object tables, there are $n+1$ distinct copies of the Shared Area, where n is the number of participants of the session. As the title implies, the Coshed/SAS system has client/server architecture or, more specifically, FRSSA. Messgaes between Coshed and SAS are handled by SAC(Shared Area Client) module of Coshed and SAS. Concurrency control is primarily based on locking. Transactions from clients are either sent to SAS or broadcasted to other clients. Upon reception, SAS processes the messages and broadcasts to all clients including the requester. Since each client has its local replicated copy of object table, object table update messages are immediately applied to the table. As consensus among replication servers is not required, FRSSA allows faster transaction processing. Intensive read operations such as window redrawing, are instantaneous because of the

existence of local object tables. Messages of user interface related operations such as telepointing are broadcaseted by the client to all clients.

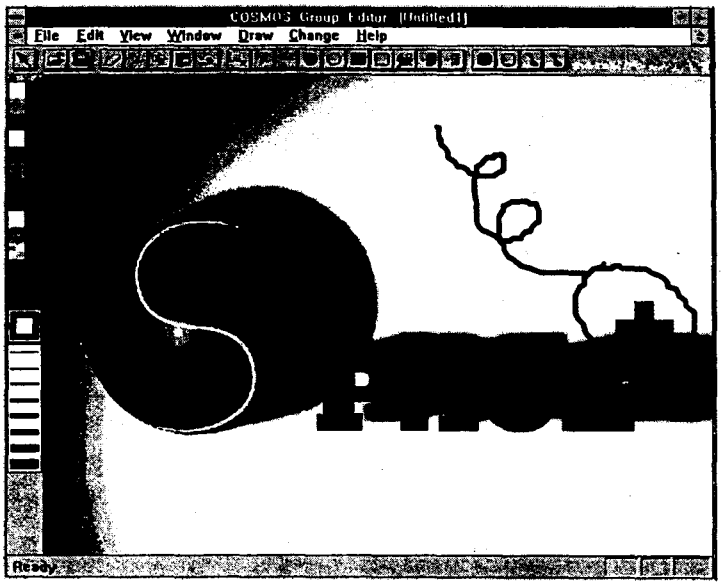

Figure 2: Coshed/SAS, a graphical shared editor

4 Conclusion

In this paper we reviewed the inherent problem of concurrency control in groupware applications and its effects on user interfaces. The traditional concurrency control algorithms are divided into three categories, *locking, timestamp ordering,* and *optimistic concurrency control.* Benefits and caveats of each algorithm are presented and the algorithms are also reviewed in distributed systems as well as groupware application systems. In Section 3, we compared the three algorithms in terms of users' feel and degree of concurrency and found that both ends could not be achived simultaneously to full extents. The optimistic concurrency control algorithms, for example, guarantees the highest degree of concurrency but recovery operations necessary in case of conflicting transactions can be unnatural to users who are acustomed to cooperation transparent applications. We also considered the effects of the geometry of the system and the choice of replication scheme on the system performance. By the geometry we refer mean number of servers employed. Server-based system can efficiently reduce the number of connections necessary to complete transaction requests and it is easy to implement concurrency control

algorithms. Replicated(serverless fully replicated) systems, on the other hand, enhances read operations and concurrency of user inputs but makes implementation of concurrency control algorithms difficult. We can summarize the paper by presenting two objectives and three factors that affects system design.

o natural user interface	i concurrency control algorithm
o Concurrency of user input	i geometry (number of servers)
	i replication

We found that FRSSA (Fully Replicated Single Server Architecture) is one of the most efficient architectures and an example of a graphical editor, Coshed/SAS, based on FRSSA and a locking based concurrency control algorithm is reviewed.

References

[1] J.C. Lauwers and K. A. Lantz. Collaboration awareness insupport of collaboration transparency: requirements for the next generation of shared window systems, *CHI 1990 Proceedings*, (CHI, 1990) pp. 303-310.

[2] C. A. Ellis and S. J. Gibbs, Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, (1989) pp. 399-407

[3] I. Grief, R. Seliger and W. Weihl, Atomic data abstractions in a distributed collaborative editing system. In *Proceedings of the 13th Annual Symposium on Principles of Programming Languages,* (1986) pp. 160-172.

[4] A. Karsenty and M. Beaudouin-Lafon, An algorithm for distributed groupware applications. In *Proceedings of the 13th International Conference on Distributed Computing Systems ICDCS'93,* (1993)

[5] M. Knister and A. Prakash, Issues in the design of a toolkit for supporting multiple group editors. *Computing Systems (The Journal of the Usenix Association)*, 6(2), (1993) pp. 135-166.

[6] R. E. Newman-Wolfe and H. K. Pelimuhandiram, MACE: A Fine Grained Concurrent Editor, In *Proceedings of the ACM COCS Conference on Organizational Computing Systems*, (1991) pp. 240-254.

[7] G. Coulouris, J. Dollimore and T. Kindberg, Distributed Systems-Concepts and Design, Addison-Wesley Publishing Company, Wokingham, 1994

[8] K. P. Eswaren, J. N. Gray, "The Notion of Consistency and Recovery in a Database System," *Comm. of ACM*, vol. 19, no. 11. (1976) pp. 624-633.

[9] P. A. Bernstein, N. Goodman, Timestamp Based Algorithms for Concurrency Control in Distributed Database Systems, In *6th International Conference on Very Large Databases*, (1980) pp. 285-300.

[10] H. T. Kung and J. T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM TODS*, vol. 6, no. 2, (1981) pp.213-226

[11] J. Huang, J. A. Stankovic, K. Ramamritham and D. Towsley, "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes," In *19th International Conference on Very Large Data Bases*, (1991) pp.35-46.

[12] S. R. Ahuja, J. R. Ensor, and S. E. Lucco, "A comparison of applications sharing mechanisms in realtime desktop conferencing systems." In *Proceedings of the ACM COIS Conference on Office Information Systems,* Boston, April 25-27 (1990) pp. 238-248.

[13] S. Greenberg, "Sharing views and interactions with single-user applications." In *Proceedings of the ACM COIS Conference on Office Information Systems,* Boston, April 25-27 (1990) pp. 227-237.

[14] J. F. Patterson, R. D. Hill, S. L. Rohall, and W. S. Meeks, "Rendezvous: An architecture for synchronous multi-user applications." In *Proceedings of the ACM CSCW Conference on Computer -Supported Coopertive Work,* Toronto, November 7-10 (1990) pp. 273-280.