

Efficient Recovery Algorithm in Real-Time and Fault-Tolerant Collaborative Editing Systems

Xiao Qin Chengzheng Sun
School of Computing and Information Technology
Griffith University
Brisbane, Queensland 4111, Australia
xqin@cit.gu.edu.au, C.Sun@cit.gu.edu.au

ABSTRACT

This paper discusses the fault-tolerant issues in real-time collaborative editing systems. In order to make the real-time collaborative systems more reliable, an efficient recovery algorithm is presented. A crashed client site can be recovered by transmitting the system's final state from the server. If the volume data associated with final state is huge, the recovery latency becomes significant large. We investigate a new approach, in which each client site maintains a local final state that is generated periodically. Thus, if a failure occurs in the client or links, the client can rejoin the collaborative editing systems by loading the local final state instead of obtaining the state from the remote server that may result in a noticeable delay. The key point in our approach is the consistency between the local state and remote state, which is maintained properly in our algorithm. Interval time between a client join and leave the system is an important metric that address in our paper. We derive the equations of this interval time to measure the performance of the recovery algorithm. Regarding to this interval time, the performance of the system can be enhanced by determining an optimal frequency of generating local final state.

Keywords

Collaborative editors, fault-tolerant, real-time, consistency maintenance, checkpoint, final state

INTRODUCTION

Groupware systems allow physically dispersed teams to collaborate over common tasks over distance and/or time [4][12]. In a real-time groupware system, all users are required to be present at their respective sites at the same time, whereas a non real-time groupware system allows users to work on common tasks at different times. Real-time collaborative editing systems, that enable groups of geographically distributed users to simultaneously view and edit shared document, make the groupware applications more practical [3][14][19][21][23]. This is even more pronounced if users can use real-time collaborative editing systems on the Internet [23].

In real-time collaborative editing systems, good responsiveness, supporting unconstrained collaboration and tolerant failed processes are main issues. Hence, if a real-time collaborative editor is to be effectively used over the Internet, the system should tolerant the client and link failures, for the quality of the Internet are unpredictable [16]. There are two main approaches to improving the fault tolerance: replication [6][13] and persistence [17]. Components are replicated to make the systems fault-tolerant by ensuring that all replicates process the same messages in the same order. If any one of them fails, others will still be able to continue. Persistence-based solutions rely on *checkpointing*, which can recover the failures by periodically saving the states of components. Checkpoint recovery may be preferable for small problems if local disks are available, but wide-area replication outperforms checkpoint recovery for larger-grain problem [22].

In a real-time collaborative editing system, the clients are able to rejoin the system in the presence of the client or link failures. One basic requirement is that the existing user can continue their work while a new crashed client join the group again. Thus, the group's current status should transfer to the new client even in the presence of any failures. From the client's aspect, it can rejoin the collaborative editing system without start from the very beginning. Normally speaking, starting from the scratch can result in a substantial delay, that is unnecessary is an efficient approach is applied.

In this paper, we devise a new, efficient approach to support crash recovery in the real-time collaborative editing systems. In regarding to the fault-tolerant support for server in the system, we have developed the *primary-backup server* to tolerate the single server failure [23]. If the primary server is crashed, the backup server will automatically continue to server the clients without restarting the server. This paper introduces the notion of *local final state*, which records the final state of the client. In order to protect the clients from crash, local final state is stored on permanent storage at each client site. If a client gets disconnected because of the server crashes or link failures, the client are able to rejoin the collaborative editing system by loading the local final state. In order to synchronize with the current state of the whole system, the

server also will resend some operations according to this local final state. How to determine the operations that should be resent by the server is the main issue discussed in this paper.

The rest of the paper is organized as follows. In Section 2, the model and assumptions are presented. Algorithms are proposed in Section 3. The performance analysis is presented in Section 4. In Section 5, we present the related work, and section 6 summarizes the contributions of this paper and suggests future directions of this work.

SYSTEM MODEL

The real-time collaborative editing system is modeled by a pair $CES = \langle S, C \rangle$. S is a finite set of sites $S = \{s_0, s_1, s_2, \dots, s_n\}$, where s_0 is the server and other sites are clients. C is a finite set of channels, $C = \{c_1, c_2, \dots, c_n\}$, where c_i is a point-to-point channels that connect site s_i ($i \neq 0$) and s_0 . s_i 's execution is a sequence of operations which includes a the remote operations from other sites. Site s is denoted as, $s_i = \langle LHL_i, L_i, \xi_i, LFS_i \rangle$, where LHL_i is the local history list of operation that executed on site s_i , L_i is the locking table and ξ_i is the state of site, we will discuss the state of the site in detail later. LFS_i is the Local Final State that is generated periodically and stored on permanent storage, when the client site crashed or the link fails, LFS_i will be used to initialize the site. LFS is model as a tuple, $LFS = \langle \xi, co, C, L \rangle$, where ξ is the site state, co is the last operation associated with LFS , C is the content of the document, L is the locking table. Remote Final State (RFS) has the same structure as LFS .

Before giving the "happened before (\rightarrow)" relation, we define the operation formally as follows.

Definition 1 Operation o is a tuple, $o = \langle \gamma, \tau, \phi, \Delta \rangle$, where γ is the site's identifier that originally generates operation o , τ is the time at which site s_γ generates and executes the operation, ϕ is the type of the operation, and Δ is the data associated with this operation.

Definition 2 Assume site i generates and executes operation o , and this operation is sent via the channel to site j , and site j will execute it as a remote operation. A remote operation ro is a tuple, $ro_j^i(o) = \langle \tau, \phi, \Delta \rangle$, where τ is the time at which site s_j receive the operation, ϕ is the type of the operation, and Δ is the data associated with this operation.

The "happened before (\rightarrow)" relation between two operations in CES is defined as follows, which is similar as that in [8].

Definition 3 $o_i \rightarrow o_j$ if and only if:

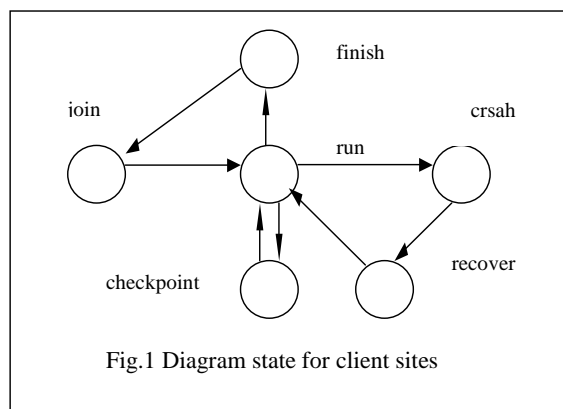
- (1) $o_i.\gamma = o_j.\gamma$ and $o_i.\tau < o_j.\tau$;
- (2) $o_i.\gamma \neq o_j.\gamma$, $ro_j^k(o_i).\tau < o_j.\tau$, where $k = o_i.\gamma$;
- (3) There exists an operation o_k , such that $o_i \rightarrow o_k$, $o_k \rightarrow o_j$.

Definition 4 Operation o_i and o_j is independent if only if

neither $o_i \rightarrow o_j$, nor $o_j \rightarrow o_i$, which is defined as $o_i \parallel o_j$.

Definition 5 Let $ol_j(o_i)$ be the last operation for o_i at site s_j , it satisfies: $ol_j(o_i) \in LHL_j$ and $ol_j(o_i) \rightarrow o_i$ and $\forall o_k \in LHL_j$, $o_k \rightarrow o_i$: $o_k \rightarrow ol_j(o_i)$.

Each site s_i maintains a status ξ_i that is one of the following: *join*, *run*, *checkpoint*, *recover*, *crash* and *finish*. The client site starts by loading the Local Final State (LFS) from the local permanent storage. If no LFS is available, the state will be initialized to be *join* and remain in this state until it receives the Remote Final State (RFS) from the server and execute operations according to the RFS. On the other hand, if the client site has a LFS, how to setup client site is depend on the state in LFS. The state should be either *finish* or *run*. Under this situation, if the state is *finish*, the state will changed from *finish* into *join* and the client will obtain the RFS from the server, the state will change from *join* into *run* after the client execute operations associated with RFS. Or, if the state is *crash*, it will change into *recover* and begin loading all the data in LFS, after finish obtaining LFS, the state will turn into *run*. The user's interface will not be enabled until the state of the client becomes to *run*. The state diagram is described in Figure 1.



RECOVERY ALGORITHM

This session presents the recovery algorithm according to our new approach. The topology of the collaborative system we study is a start-like network, in which the server is the central part of the system and each client connects to server with a channel respectively.

The server does not generate any editing operations, but receive operations from client sites, maintains a remote final state of the system and propagate the operations to other client sites. The server also manages the memberships of the sites in the collaborative editing systems. The server itself, of course, may crash, even server can be made more reliable and better controlled than client sites. If the server is down, no new client can join the collaborative editing system and no proper notification can be broadcast to other sites. To improve the dependability of the server in the runtime environment, especially in the Internet environment, the primary-backup fault-tolerant technique, which is a useful approach [13], is employed in server. Two server sites are linked via Internet or LAN, one

serves as the primary and another is used for backup purpose to tolerate the single server failure. The backup server also maintains the remote final state of the system but it has no responsibility to propagate the operations from client sites. As a result, it can take the place of the primary server automatically to manage the collaborative editing system when the primary server is crashed.

The server starts by loading remote final status that stored on the permanent storage, then the algorithm initializes the remote history list (RHL) to empty. After the initialization phase, the server will wait for messages from client sites and processes according to the message it receives. If the server receives a join message, it sends back the remote final state to the client that sends this join request. After sending the RFS, this client site is added to the set of sites S as a member of the collaborative editing system, hence this client site can receive the operations broadcast from the server. If a client send a finish request, the server deletes it from the group by removing it from set of sites S .

If the server receives a RECOVER message from s_i , it means that s_i crashed just now, and rejoins the collaborative system by loading its local final state. If after the creation of the LFS_i at site s_i , an existing client generates, at least, one editing operation, then the local final state is inconsistent with remote final state at server. In order to maintain consistency of the collaborative editing system, the server must send all the operations that were generated after the creation of the LFS_i . Assume co is the operation at site s_i that creates LFS_i , the operations being sent is in which is defined as follows:

$$LOS(co, RHL) = \{o \mid ol_0(co) \rightarrow o, \text{ where } o \in RHL\} \quad (1)$$

If the message contains the editing operations from the clients, the server transforms the operation according to the history list, executes the transformed operation and append it into history list. The server also sends the operation to other client sites in the system. The description of the algorithm in server is given below.

Algorithm in Server Site:

```

Load RFS; RHL ← ∅;
WHILE ( $s_0$  is active) DO
    Wait and receive message  $m$  from other sites;
     $mt \leftarrow \text{Get\_Type}(m)$ ;  $i \leftarrow \text{Get\_SenderId}(m)$ ;
    SWITCH ( $mt$ )
    CASE JOIN message:
        Send RFL to site  $s_i$ ;
         $S \leftarrow S + \{s_i\}$ ;
    CASE FINISH message:
         $S \leftarrow S - \{s_i\}$ ;
    CASE RECOVER message:
         $co \leftarrow \text{LastOperation}(m)$ ;
        Send all  $o \in LOS(co, RHL)$  to site  $s_i$ ;

```

CASE other operation:

```

 $o \leftarrow \text{Get\_Operation}(m)$ ;  $eo \leftarrow \text{transform}(RHL, o)$ ;
Execute  $eo$ ;  $RHL \leftarrow RHL + o$ ;
Broadcast  $o$  to all  $s_j \in S$ , where  $j \neq 0, j \neq i$ ;

```

END WHILE

END

If a collaborative editing system is to be effectively used, it should allow clients sites to tolerate client and link failures and server crashed. This is even more pronounced when the collaborative editing system is used over a wide area network such as the Internet and World Wide Web, where the quality of the networking and computing resources are unpredictable. Since the server crashed is tolerated by primary-backup model, a new approach of recovery for the client and link failure is devised. When client or link failures occurs, the client should be allowed to rejoin the system without starting over from scratch. In the traditional way, the server transmits the system's state to the client sites. If the shared document is very large, the communication delay for the state transmission is significant, and the users may impatient with this transmission delay. Therefore, it is very important to minimize delay for recovering shared state when crashed client rejoin the system.

In our approach, we reduce the state transmission delay by loading the system's state from local permanent storage instead of the remote final state. As discussed before, in case that some operations are generated after the creation of the local final state LFS_i , LFS_i at the client site s_i becomes inconsistent with remote final state at server. So, consistency control must be considered during the recovery procedure. If the state is *recover*, the client site first loads the state from LFS_i . Then, the site sends a RECOVER message, which contains the operation co that generates this LFS_i , to the server site. When the server receives such co , it will send all the operations, which were generated after the creation of the LFS_i , back to site s_i .

If the current state is *join*, the client site send a JOIN message to the server, then wait for RFS sent from the server. After RFS is received and the site is initialized according to the RFS, the state of the site changes into *run*. If the state is *checkpoint*, it creates a LFS and stores it on the local permanent storage. After LFS stored, the site changes its state to *run*. In case that the state is *finish*, it generates a local final state and saves it, followed by releasing all the locks that generated by this site and sending a FINISH message to the server.

In most cases, the site is in *run* state. The site waits for operations and processes according to each operation received. If the operation is the local finish operation, the state is changed from *run* into *finish*. If the operation is other kinds of local operations, the site executes the operation, appends the operation into its LHL and sends it to the server, so that this local operation can be propagated to other sites in the system. If the operation is a remote operation generated at other sites, it is transformed before

being executed [21]. When the transformed remote operation is executed, it is appended in LHL. The algorithm in client sites is outlined as follows:

Algorithms in Client Site:

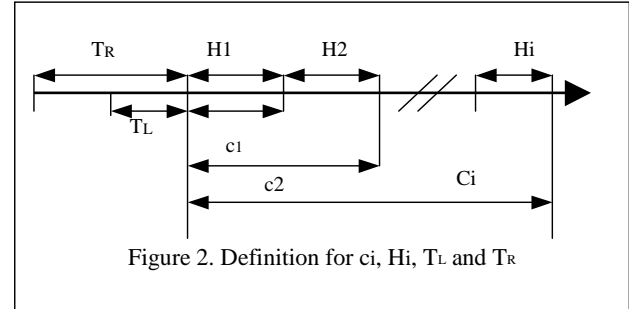
```

Load LFSi;  $\xi_i \leftarrow \text{Get\_S}(\text{LFS}_i)$ ;
IF ( $\xi_i = \emptyset$  OR  $\xi_i = \text{finish}$ ) THEN  $\xi_i \leftarrow \text{join}$ ;
ELSE  $\xi_i \leftarrow \text{recover}$ ;
WHILE ( $s_i$  is active) DO
  SWITCH ( $\xi_i$ )
  CASE join:
    Generate JOIN message and send it to server;
    Wait for response message  $r$ ;
     $\text{RFS}_i \leftarrow \text{Get\_RFS}(r)$ ;  $\text{LFS}_i \leftarrow \text{RFS}_i$ ;
    Initialize site  $s_i$  with  $\text{RFS}_i$ ;  $\xi_i \leftarrow \text{run}$ ;
  CASE recover:
     $\text{LFS}_i \leftarrow \text{Load\_LFS}()$ ;
     $co \leftarrow \text{Get\_LastOperation}(\text{LFS}_i)$ ;
    Generate a recovery message that contains  $lo$ ;
    Send recovery message to server;  $\xi_i \leftarrow \text{run}$ ;
  CASE checkpoint:
     $lo_i \leftarrow \text{Get\_LO}(\text{LHL}_i)$ ;
     $\text{LFS}_i \leftarrow \text{Create\_LFS}(\xi_i, co_i, C_i, L_i)$ ;
    Store  $\text{LFS}_i$  on the permanent storage;  $\xi_i \leftarrow \text{run}$ ;
  CASE run:
    WHILE ( $\xi_i = \text{run}$ ) DO
      Receive operation  $o$ ;
      IF  $o$  is local operation THEN
        IF  $o$  is a finish operation THEN  $\xi_i \leftarrow \text{finish}$ ;
        ELSE Execute  $o$ ;  $\text{LHL}_i \leftarrow \text{LHL}_i + o$ ;
        Send operation  $o$  to server, so that it can
          broadcast it to other sites in the system;
        ELSE /* This is a remote operation */
           $eo \leftarrow \text{transform}(\text{LHL}_i, o)$ ;
          Execute  $eo$ ;  $\text{LHL}_i \leftarrow \text{LHL}_i + o$ ;
        END IF
      END WHILE
  CASE finish:
     $\text{LFS}_i \leftarrow \text{Create\_LFS}(\xi_i, co_i, C_i, L_i)$ ;
    Store  $\text{LFS}_i$  on the permanent storage;
    Release all locks that generated by site  $s_i$ ;
    Generate FINISH message and send it to server;
     $\xi_i \leftarrow \text{finish}$ ;
END WHILE;

```

PERFORMANCE ANALYSIS

Assume that when client site leaves the collaborative editing system successfully, it had created m checkpoints. We derive the expected interval time between a client join and leave the system, since this metric is a good one to evaluate the performance of the algorithm.



As displayed in figure 2, P_i ($i \in [2, m]$) represents the execution time on client site, it is the nominal measured in CPU cycles between $(i-1)$ th and i th checkpoints, P_i denotes the interval between the beginning of the client and its first checkpoint without any failures. The sum of the execution time is defined as $P = \sum_{i=1}^m P_i$.

Let c_i ($i \in [1, m]$) be the execution time from the beginning of the client site to the i th checkpoint in presence of the client or link failures. Let C_i be the expected value of c_i , $C_i = E(c_i)$, thus, the expected interval time of a client between join and leave the collaborative editing system is $C_m = E(c_m)$.

The client and link failures can be recovered by loading local final state or remote final state, let p and q be the probability of recovering the client by loading LFS and RFS, respectively, it is clear that $p + q = 1$. Suppose that time overhead for loading LFS and RFS are T_L and T_R , respectively, and T_C represents the overhead for creating and saving a LFS as the checkpoint. $f_i(t)$ ($i \in [2, m]$) denotes the probability of a client/link failure in t unit of time from the time of the $(i-1)$ th checkpoint. $f_1(t)$ is the failure probability from the very beginning. We have,

$$c_i = \begin{cases} P_i & \text{with probability } 1 - f_1(P_i) \\ P_i + T_L + c_i & \text{with probability } p \times f_1(P_i) \\ P_i + T_R + c_i & \text{with probability } q \times f_1(P_i) \end{cases} \quad (2)$$

Let H_i be the time interval between $(i-1)$ th and i th checkpoint. Thus, for $i \in [2, m]$,

$$c_i = c_{i-1} + H_i + T_C \quad (3)$$

$$H_i = c_i \quad (4)$$

$$H_i = \begin{cases} P_i & \text{with probability } 1 - f_i(P_i) \\ P_i + T_L + c_i - c_{i-1} & \text{with probability } p \times f_i(P_i) \\ P_i + T_R + c_i - c_{i-1} & \text{with probability } q \times f_i(P_i) \end{cases} \quad (5)$$

We could derive the C_i from the above equations as follows,

where $i \in [2, m]$,

$$C_i = C_{i-1} + P_i + [(T_L p + T_R q) f_i(P_i) + TC] / (1 - f_i(P_i)) \quad (6)$$

We can obtain the expected interval time of the client between join and leave the system by applying the above equation $m-1$ times,

$$C_m = \sum_{i=1}^m \{ [P_i + (T_L p + T_R q) f_i(P_i)] / [1 - f_i(P_i)] \} \quad (7)$$

In order to minimize C_m to improve the performance of the system, the checkpointing frequency should be determined properly. The value of m that minimizes the equation (7) is an optimal one.

Let $C^L(P, k)$ denote the execution time of the client in the presence of up to k recovering by loading LFS, let $p_i^{L,S}$ and $p_i^{L,U}$ be the probability of the i th LFS approach becoming successful and unsuccessful, respectively, where $p_i^{L,S} + p_i^{L,U} = 1$. $C_1^L(P, k)$ is given as below,

$$\begin{aligned} C^L(P, k) &= p_i^{L,S} + \{ P + p_i^{L,U} p_2^{L,S} (T_L + P) / [1 - f_1(P)] \} + \\ &\quad \{ 2P + p_i^{L,U} p_2^{L,U} p_2^{L,S} (T_L + P) / [1 - f_1(P)] \} + \dots + \\ &\quad \{ kP + \prod_{i=1}^k p_i^{L,U} p_{k+1}^{L,S} (T_L + P) / [1 - f_1(P)] \} \\ &= \sum_{j=0}^k [jP + (T_L + P) / (1 - f_1(P))] \prod_{i=1}^j p_i^{L,U} p_{j+1}^{L,S} \quad (8) \end{aligned}$$

LFS is a efficient method to recover the temporary failures in client and links, if the permanent failure occurs and LFS stored on local storage is missing, client should rejoin the system by loading RFS. Similarly, let $C^R(P, k)$ be the execution time of the client in the presence of up to k recovering by loading RFS, and $p_i^{R,S}$ and $p_i^{R,U}$ be the probability of the i th LFS approach becoming successful and unsuccessful, where $p_i^{R,S} + p_i^{R,U} = 1$. $C^R(P, k)$ is expressed as follows,

$$\begin{aligned} C^R(P, k) &= \sum_{j=0}^k [jP + (T_R + P) / (1 - f_1(P))] \prod_{i=1}^j p_i^{R,U} p_{j+1}^{R,S} \quad (9) \end{aligned}$$

In our recovery approach, let V_L be the volume of data needs to be transmitted from server to rejoining client for purpose of consistency maintenance. Assume that LFS is the local final state to be loaded, $o_L = \text{LFS.co}$, V_L can be express as follows, where LOS is define in equation (1), and V_o denotes the volume of data associate with operation

$$o, V_L = \sum_{o \in \text{LOS}(o_L, \text{RHL})} V_o$$

If the client is recovered by obtained RFS from the server, the volume of data needs to be sent from the server to the client is modeled as V_R . V_R is given as follows,

$V_R = V_\xi + V_{co} + V_C + V_L$, where V_ξ, V_{co}, V_C , and V_L are volume of data associated with the site state, the last operation, content of the document and the locking table, respectively. To minimize the overhead in the server, our

approach is used only if $V_L < V_R$, otherwise RFS approach outperforms our algorithm in term of the recovery overhead for server.

RELATED WORK

Many systems have been proposed to support collaborative editing. Collaborative editing systems can be classified into two categories: Real-time and non real-time [15]. Real-time collaborative editing systems are most effective during the initial and integration/reviewing stages of collaborative authoring, whereas non real-time collaborative systems can improve the efficiency of collaboration in authoring team, especially document management. If the authoring team adopts explicit task-related collaboration strategies, or human protocols, to guide the collaboration process, two kinds of collaborative editing systems can be used more effectively.

Non real-time collaborative editing systems have shared documents that can be accessed and locked separately. A shared repository, such as distributed file system, serves as the infrastructure for many non real-time collaborative systems [10][11]. In real-time collaborative editing systems, multiple users are allowed to concurrently and freely edit any part of the document simultaneously. The response to local user actions is quick, and the latency for reflecting remote user actions is depend on external communication latency, which is usually slow [21]. REDUCE [23] and SASSE [2] are good examples for real-time collaborative systems.

Most the research works in real-time collaborative editing systems focus on consistency maintenance [20] [21], user intention preservation [8], group undo [18], and group awareness [6][24], fault-tolerance and reliability issues, however, have not been studied deeply. If the real-time collaborative editing system is to be efficiently used over a wide area network, the fault-tolerant issues must be take into account, for the reason that wide area network are usually unreliable [16]. If group communication subsystems are designed and implemented properly, they can provide an infrastructure for building distributed and reliable services on top of their message broadcasting and membership services [1][9]. The drawback of these systems is that they do not directly manage groups' shared application state and transfers groups' state to new clients. Paper [16] employs the notion of a stateful group communication, and provides more practical fault-tolerant services for collaborative system. Since these systems are designed for general purpose, they do not study the concurrency control issues, which is one of the most import points in collaborative editing systems.

Paper [7] presents the requirements for the collaborative editor, and proposes a model, in which fault-tolerant is mentioned. The model is able to connect to the session even in the presence of network failures by trying to guarantee that there is always a passive copy on a

highly available host. This technique is also discussed in paper [1]. But they do not consider the consistency maintenance, which is fully taken into account in our approach. PREP [10] is a collaborative writing system that uses the concept of flexible diff-ing for reporting differences between versions of texts. It is clear that this system supports the crash recovery automatically. The major difference between this approach and our algorithm is that, it is only suitable for non real-time collaborative editing systems, but our algorithm is devised for real-time collaborative editing systems.

CONCLUSIONS

In this paper we attempted to address fault-tolerant issues in real-time collaborative systems. An efficient recovery algorithm is presented to make the real-time collaborative systems more reliable. Traditional way to recover a crashed client site is transmitting the system's final state, which includes the content of the document and locking table, from the server site. But if the volume of data of the final state is huge, the recovery latency becomes significant large, which may make use feel impatient. In our new approach, each client site maintains a local final state, which is generated periodically. As a result, if a failure occurs in the client or links, the client are able to rejoin the collaborative editing systems by loading the local final state instead of obtaining the state from remote server that may result in a noticeable delay. The consistency between the local state and remote state is maintained in our algorithm. Interval time between a client join and leave the system is an important metric that addresses in our paper. We derive an equation to determine such interval time. Regarding to this interval time, the performance of the system can be enhanced by determining an optimal frequency of generating local final state.

Future studies in this work focus on performance evaluation for our new approach. We will find out how factors, affect the performance of the system. These factors include, the data volume associated with final state of the system and the frequency of generating local final state. The new approach proposed in this paper can also be applied in other sorts of collaborative systems. We also plan to investigate a fault-tolerant mechanism to provide the basic fault-tolerant services in collaborative systems.

ACKNOWLEDGMENTS

The authors wish to thank Lingzhong Zhou and Haifeng Shen for their insightful comments and suggestions. The work reported in this paper was supported in part by an ARC (Australia Research Council) Small Grant and a Large Grant (A49601841).

REFERENCES

- [1] Y. Amir, D. Dolev, S.Kramer, and D. Malki. Transis: A Communication Sub-System for High Availability. Technical Report TR CS91-13, Computer Science Department, Hebrew University, April 1992.
- [2] E.E. Beck and V.M.E. Bellotti. Informed Opportunism as Strategy: Supporting Coordination in Distributed Collaborative Writing. *In Proceedings of 3rd European Conference on Computer Supported Cooperative Work*, Milan, Italy, pp.233-248, 1993.
- [3] David Chen and Chengzheng Sun. A distributed algorithm for graphic objects replication in real-time group editors. *In Proceedings of the international ACM SIGGROUP conference on Supporting group work*, Phoenix, AZ USA, pp.121-130, 1999.
- [4] C.A.Ellis. Coordination Technology, Trends in CSCW, Michel Beaudouin-Lafon (ed.), Wiley, 1997.
- [5] P. Godefroid, J.D. Herbsleb, L.J. Jagadeesan, and Du Li. Ensuring Privacy in Presence Awareness Systems: An Automated Verification Approach. *In Proceedings of ACM 2000 Conference on Computer Supported Cooperative Work*, December, Philadelphia, Pennsylvania, USA, 2000.
- [6] H. Higaki, K. Tanaka, and M. Takizawa. Protocol for Pseudo-Active Replication in Wide-Area Networks. *In Proceeding of the 2nd International Workshop on Network-Based Information Systems (NBIS)*, pp.678-682, 1999.
- [7] M. Koch. Design Issues and Model for a Distributed Multi-user Editor. *Computer Supported Cooperative Work*, 3(3-4), pp.359-378, 1995.
- [8] Du Li, Limin Zhou, and Richard R. Muntz. A new paradigm of user intention preservation in realtime collaborative editing systems. *In Proceedings of the Seventh International Conference on Parallel and Distributed Systems*, Iwate, Japan, July, 2000.
- [9] S. Mishra, L.L. Peterson, and R.D. Schlichting. Consul: A Communication Substrate for Fault-Tolerant Distributed Program. *Distributed Systems Engineering Journal*, 1(2), pp.87-103, Dec. 1993.
- [10] C.M. Neuwirth, R. Chandhok, D.S.Kaufer, P.Erion, J.H. Morris, and D.Miller. Flexible Diff-ing in a Collaborative Writing System. *In Proceedings of 4th International Conference on Computer Supported Cooperative Work*, Toronto, CA, pp.147-154, 1992.
- [11] F. Pacull, A. Sandoz, and A. Schiper. Duplex: A Distributed Collaborative Editing Environment in Large Scale. *In Proceedings of International Conference on Computer Supported Cooperative Work*, Chapel Hill, NC, pp.165-173, October 1994.
- [12] A. Prakash, H.S. Shim, and J.H. Lee. Issues and Tradeoffs in CSCW Systems. *IEEE Transactions on Data and Knowledge Engineering*, Jan.-Feb., 11(1), pp. 213-227, 1999.
- [13] Xiao Qin, Z.F. Han, H. Jin, L.P Pang and SL Li, Real-time Fault-tolerant Scheduling in Heterogeneous Distributed Systems, *Proceeding of the 2000 International Workshop on Cluster Computing-Technologies, Environments, and*

Applications (CC-TEA'2000), Las Vegas, Nevada, USA, June, 2000.

- [14] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenbauser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. *In Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pp. 288-297, 1996.
- [15] M. A. Sasse and M. J. Handley. Collaborative Writing With Synchronous and Asynchronous Support Environments. In Rada, R. [Ed.]: *Groupware and Authoring*, Academic Press, pp 205-218, 1996.
- [16] H. S. Shim, A. Prakash. Tolerating Client and Communication Failures in Distributed Groupware Systems. *In Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pp.221-227, West Lafayette, Indiana, USA, 1998.
- [17] K. F. Ssu, B. Yao and W. K. Fuchs. An Adaptive Checkpointing Protocol to Bound Recovery Time with Message Logging. *In Proceeding of the 18th IEEE Symposium on Reliable Distributed Systems*, Lausanne, Switzerland, October, 1999
- [18] C. Sun. Undo any operation at any time in group editors. *In Proceedings of ACM 2000 Conference on Computer Supported Cooperative Work*, December, Philadelphia, Pennsylvania, USA, 2000.
- [19] C. Sun and C.A. Ellis. Operational transformation in real-time group editors: Issues, algorithms, and achievements. *In Proceedings of ACM Conference on Computer Supported Cooperative Work*, Seattle, USA, November 1998.
- [20] C. Sun and R. Sasic. Consistency maintenance in Web-based real-time group editors. *In Proceedings of the 19th IEEE International Conference on Distributed Computing Systems.*, Austin, TX, USA, pp. 15-22, May 31- June 4, 1999.
- [21] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality-preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63-108, March 1998.
- [22] J. B. Weissman. Fault Tolerant Wide-Area Parallel Computing. *Fault-tolerant Wide-Area Computing. Workshop on Fault-Tolerant Parallel and Distributed Systems FTPDS '00*, April 2000.
- [23] Y. Yang, C Sun, Y.C. Zhang, and X.H. Jia. Real-Time Cooperative Editing on the Internet. *IEEE Internet Computing*, pp.18-25, May/June 2000.
- [24] Y. Yokota, H. Tarumi, and Y. Kambayashi. Extended Awareness Support for Cooperative Work in Non-WYSIWIS Condition. *In Proceedings of International Computer Science Conference*, Springer-Verlag, 1999.