Recovery Support for Internet-based Real-Time Collaborative Editing Systems

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

Abstract

For reliability, recovery support must be employed in the Internet-based real-time collaborative editing systems. This paper describes a recovery scheme in which each site maintains a local document state (LDS) that is generated periodically. Thus, if a failure occurs in the Internet links or at the site, the site can rejoin the collaborative editing system by loading the LDS instead of obtaining the state from the remote sites. This is a much faster process than the traditional approach the recovery of regaining the system's document state from other peer sites. Consistency between the local state and remote state during the recovery procedure is maintained in a recovery algorithm for which the proof is provided. The performance of our recovery scheme is assessed by generating the elapsed time between a failed site joining and leaving the systems.

1. Introduction

Using real-time collaborative editing systems, groups of geographically distributed users can view and edit simultaneously shared documents. Real-time collaborative editing systems are important groupware applications [3][14][19][21][23]. It is important for these systems to have a good responsiveness, support of unconstrained collaboration, and are tolerant of failed processes. A real-time collaborative editor system on the Internet must tolerate site and link failures [16]. Two main fault tolerant techniques are replication [6][13] and persistence [17]. With replication, the hardware and software components replicated process the same messages in the same order. If any one fails, others are still able to continue. Persistence-based solutions rely on *checkpointing* whereby during the normal execution, application states are periodically saved on a stable storage and used during the recovery to rollback to an earlier consistent state.

Traditionally, recovery of a failed site has been implemented by regaining the system's document state from other peer sites. However, the recovery latency can be significantly long if the volume of data associated with the document state is large. Substantial delays can be reduced if recovery does not start from scratch. A failed site can rejoin the collaborative editing system without starting from the very beginning if an efficient approach is used.

We investigated an approach in which each site maintains a LDS that is generated periodically. Thus, if a failure occurs in the Internet links or at the site, the site can rejoin the collaborative editing system by loading the LDS instead of obtaining the state from the remote sites, a much faster process. During the recovery procedure, the consistency between the local state and remote state is maintained in a recovery algorithm.

The rest of the paper is organized as follows. In Section 2, we present the system model. Fault-tolerant algorithms and the proof of the correctness are given in section 3. The performance analysis of the algorithm is presented in section 4. Section 5 outlines the related work, and section 6 summarizes the contributions of this paper and suggests future directions of our research.

2. System Model

The Internet-based real-time collaborative editing system is modeled by a pair CES = <S, C>. S is a finite set of sites S = $\{s_1, s_2, ..., s_n\}$, where s_i is a site involved in editing work. C is a finite set of channels, C = $\{c_{ij}, 1 \leq i \leq n, i < j \leq n\}$, where c_{ij} is a point-to-point channels that connect site s_i and s_j via the Internet. $s_i's$ execution is a sequence of operations which includes the remote operations from other sites. LDS_i is the LDS that is generated periodically and stored on permanent storage, when the site s_i failed, LDS_i will be used to initialize the site.

Definition 1. Given an operation O, then s(O) denotes the site at which O is generated, $e_i(O)$ represents the execution form of O at s_i, $gt_i(O)$ denotes the time when s_i generates O, and $at_i(O)$ represents the execution time of O at the remote sites s_i. It is certain that $gt_i(O) \rightarrow s(O) = i$, and $at_i(O) \rightarrow s(O) \neq i$.

Definition 2. Given two operations O_i and O_j, O_i is causal order preceding O_j, denoted by $O_i \rightarrow O_j$, iff:

(1) $s(O_i) = s(O_j) = k$, and $gt_k(O_i) < gt_k(O_j)$;

(2) $s(O_i) \neq s(O_j)$, $at_k(O_i) < gt_k(O_j)$, where $k = s(O_j)$;

(3) There exists an operation O_k, such that $O_i \rightarrow O_k$, $O_k \rightarrow O_j$.

Definition 3. Operation O_i and O_j are independent if and only if neither $O_i \rightarrow O_j$, nor $O_j \rightarrow O_i$, which is defined as $O_i \parallel O_j$.

Definition 4. An operation is associated with a context, denoted as CT_O, which is the list of operations that need to be executed to bring the document from its initial states to the states on which O is defined.

Definition 5. Given two operations O_i and O_j, associated with contexts CT_{Oi} and CT_{Oj}, O_i and O_j are context equivalent, i.e., $O_i \sqcup O_j$, if only iff, $CT_{Oi} = CT_{Oj}$.

Definition 6. Given two operations O_i and O_j, associated with contexts CT_{Oi} and CT_{Oj}, O_i is context preceding O_j, i.e., $O_i \succ O_j$, if only if, $CT_{Oj} = CT_{Oi} + [O_j]$.

Definition 7. Given two operations O_i and O_j, $s(O_i) = a$, $s(O_j) = b$, and timestamped by SV_{Oi} and SV_{Oj}, respectively [21], then O_i is total order preceding O_j, i.e., $O_i \Rightarrow O_j$, iff (1) $sum(SV_{Oi}) < sum(SV_{Oj})$ or (2) $a < b$ when $sum(SV_{Oi}) = sum(SV_{Oi})$, where $sum(SV) = \sum_{i=1}^{n} SV[i]$.

Definition 8. Let HB_i^t be the history buffer of s_i at time t. In HB_i^t, $y_i^{j, t}$ denotes the latest operation generated at site s_j, iff, $\forall O \in HB_i^t$, $O \neq y_i^{j, t}$: $s(O) = j \rightarrow (O \rightarrow y_i^{j, t})$.

Each site s_i maintains a status ξ_i that is one of the following: *join, run, checkpoint, recovery, fail and finish*. The recovery begins by loading the LDS from the local permanent storage. If no LDS is available, the state is initialized to *join* and remains in this state until it receives the *Remote Document state (RDS)* from the other sites and executes operations according to the RDS. On the other hand, if the site has a LDS then setting up the site depends on the state in LDS. If the state in LDS is *finish*, it means this site exited successfully during the last session. Then, the state changes from *finish* into *join* and the site obtains the RDS from other sites, the state changes from *join* into *run* after the site execute operations associated with RDS. If the state in LDS is run, it means that this site did not exit successfully due to the link failure or site failure. So, the state changes into *recover* followed by loading all the data in LDS. After finish obtaining LDS and receiving all missed operations entered at its own site, the state is set to *run*. The user's interface is not enabled until the state of the site is *run*. This case is described formally by theorem 2.

If the current state is *join*, the site propagates a join message to all other sites, then wait for the first remote site that reply this message. After receiving the document state from this remote site and the site is initialized, the state of the site changes into *run*. If it does not receive any reply, it assumes that it is the first one to join the system. In this case, the local document is loaded and the state is updated into *run*. If the state is *checkpoint*, it stores LDS

on the local permanent storage, and changes the state to *run*. In case that the state is *finish*, it saves LDS, followed by broadcasting a *finish* message to other sites.

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 history buffer and broadcast it to other sites. If the operation is a remote operation generated at other sites, it is transformed before being executed [21]. The state diagram is described in Figure 1.
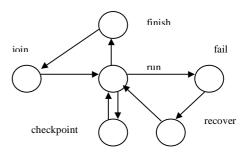

Figure1. Diagram state for sites

There are two different architectures for storing shared documents: centralized and replicated. Though centralized approach is simple, it results in the poor responsiveness, especially in the Internet environment. Replicated approach, however, exhibits a better responsive time, since each site can update the local document immediately followed by propagate to the remote sites. In replicated architecture, concurrency control to maintain consistency in replicated document is one of several essential issues. To solve the inconsistency problems, the consistency model was given in paper [21].

3. Recovery Algorithm

Before presenting the recovery algorithm, a algorithm to decide the latest operation generated at site s_j in Hb_i^t is described as follows.

Algorithm 1. $LO(HB_i^t, j)$: Given a history buffer HB_i^t at site s_i, y_i^j is the latest operation generated at s_j; and is obtained as the follows,

$\quad j \leftarrow |HB_i^t|$;

$\quad$ while $(j > 0)$ do

$\quad\quad O \leftarrow HB_i^t[j]$;

$\quad\quad$ if $s(O) = j$ then return $y_i^j = O$;

$\quad\quad$ else $j \leftarrow j - 1$;

$\quad$ end while

$\quad$ return φ;

end algorithm 1.

The following three lemmas describe the features of

two operations generated at the same site.

Lemma 1. Given two operations O_i and O_j: if $s(O_i) = s(O_j)$ then either $O_i \to O_j$ or $O_j \to O_i$.

Proof. Given $s(O_i) = s(O_j)$: if the generation of O_i happened before the generation of O_j then $O_i \to O_j$ (see Definition.2) else the generation of O_j happened before the generation of O_l and $O_j \to O_i$.

Lemma 2. Given two operations O_i and O_j: if O_i is causal order preceding O_j then O_i is total order preceding O_j and $\forall O_i, O_j: (s(O_i) \to s(O_j)) \to (O_i \Rightarrow O_j)$.

Proof given in [20].

Lemma 3. Given two operations O_i and O_j in the history buffer HB: if two operations are generated at the same site and O_i is total order preceding to O_j, then O_i is causal order preceding O_j and $\forall O_i, O_j \in$ HB: $(s(O_i) = s(O_j) \wedge O_i \Rightarrow O_j) \to (O_i \to O_j)$.

Proof. If lemma 3 is incorrect, then either $O_j \to O_i$ or $O_i \parallel O_j$. Because $s(O_i) = s(O_j)$, only $O_j \to O_i$ (lemma 1) is possible and thus $O_j \Rightarrow O_i$ (lemma 2) which is a contradiction.

Given HB_i^t and s_j, algorithm 1 determine the latest operation generated at s_j, we prove the correctness of the algorithm 1. in the following theorem.

Theorem 1. Given HB_i^t and site s_j, the operation O to be determined by algorithm $LO(HB_i^t, j)$ is y_i^j.

Proof. Since algorithm $YO(HB_i^t, j)$ scans the history buffer HB_i^t from right to left, we have $\forall O_k \in HB_i^t, O_k = HB_i^t[a], O = HB_i^t[b], O_k \neq O: s(O_k) = s(O) = j \to a < b$. Thus, $O_k \Rightarrow O$ is proved, so $O_k \to O$ (lemma 3). Hence, $\forall O_k \in HB_i^t, O_k \neq O: s(O_k) = j \to (O_k \to O)$, so $O = y_i^{j, t}$ (Def. 8).

If a link or site fails, the site is allowed to rejoin the system without starting from scratch. In our recovery approach, we reduce the state transmission delay by loading the system's state from the local permanent storage instead of the remote site. If the site is in *recovery* status, the site rejoins by loading the LDS and propagating a recovery message r, then wait for reply from other sites. Algorithm 2 outlines the procedure for a failed site rejoining the system by loading the LDS.

Algorithm 2.
Let HB_i^t be the history buffer associated with the latest checkpoint, that generated at time t.

  Local operation generation is disabled;
  $y_i^{i, t} \leftarrow LO(HB_i^t, i)$;
  for $1 \leq i \leq n$ where $i \neq j$ do
    $y_i^{j, t} \leftarrow LO(HB_i^t, j)$;
    put $y_i^{i, t}$ and $y_i^{j, t}$ into the recovery message;
    send the recovery message to site s_j;
  end for;
  while true do
    Waiting for the operations sent from peer sites:
      if O is the operation which satisfied:
        $SV_O[s(O)] \leftarrow SV_i[s(O)] + 1$ and

$SV_O[k] \leq SV_i[k]$, for all $k \in [1, n]$;
  then
  if $(s(O) = i$ and $\forall O' \in HB_i: SV_{O'} \neq SV_O)$ or $s(O) \neq i$
then
    use Undo/Transform-Do/Transform-Redo [21]
    scheme to execute O;
   end if;
    else O is delayed until two conditions are satisfied;
      if (all missed operations generated at s_i has been executed at s_i again)
    then Local operation generation is enabled;
  end while;
end algorithm 2.

In this paper, we assume that at time θ when site s_i has failed, s_i generates the latest *checkpoint* at time σ, and begins the recovery procedure by loading checkpoint and transmits the recovery message r at time γ.

It is crucial for the restored site to decide when it can start generating the operations again. In fact, the failed site s can begin operations only if it has received all lost operations generated at s_i between σ and θ from other sites. To prove the correctness of this point, we introduce theorem 2, and prove it. Before giving theorem 2, we present the property of time stamp and lemma 4.

Property 1. Let O be an operation generated at s. O is time-stamped by SV_O. After executing O at s, $SV_O[s] = SV[s] + 1$, where SV is the current local state vector.

Lemma 4. Given two operations O and O′ generated at the same site s_i, the ith sector in their time stamp are different, thus, $\forall O, O', 1 \leq i \leq n: s(O) = s(O') = i, O \neq O' \to SV_O[i] \neq SV_{O'}[i]$.

Proof. Because $s(O) = s(O') = i$, either $O \to O'$ or $O' \to O$ (lemma 1). Assume $O \to O'$ and that between O and O′, s_i generates other k-1 (k > 0) operations, thus $O \to O_{k-1} \to \ldots \to O_2 \to O_1 \to O'$, then $SV_O[i] = SV_{O_{k-1}}[i] + 1 = SV_{O_{k-2}}[i] + 2 = \ldots = SV_{O'}[i] + k$, where k > 0 (Property 1). Hence, we prove that $SV_O[i] \neq SV_{O'}[i]$. We use the same way to prove lemma 4 when $O' \to O$.

Theorem 2. Let σ, θ and γ be the latest checkpoint time, rash time and recovery time at s_i. s_i can only generate operations after time t (t > γ), when all operations generated at s_i between $\sigma < gt_i(O) < \theta$ execute at the s_i again, that is, $\forall O: \sigma < gt_i(O) < \theta \to e_i(O) \in HB_i^t$.

Proof. Suppose theorem 2 is incorrect, then s_i generates an operations O_s at time t′ > γ, when at least one operation generated at s_i between $\sigma < gt_i(O) < \theta$ does not execute at the s_i again. Thus, $\exists O: \sigma < gt_i(O) < \theta \to e_i(O) \notin HB_i^{t'}$. Let $O_1 \to O_2 \to \ldots \to O_k$ be k (k > 0) operations generated at s_i between $\sigma < gt_i(O) < \theta$, so we have $\forall 1 \leq j \leq h: e_i(O) \in HB_i^{t'}$ and $\forall h+1 \leq j \leq k\ e_i(O) \notin HB_i^{t'}$. Assume that when s_i generates the latest checkpoint at time σ, the local state vector is SV[i] = d, then after executing O_h on s_i again,

SV[i] becomes $d + h$. So, the operation $SV_{Os}[i] = d + h + 1$. The timestamp of the operation O_{h+1} that has not executed at s_i again is: $SV_{Oh+1}[i] = d + h + 1$. So we proved that $SV_{Os}[i] = SV_{Oh+1}[i]$. Since $O_s \neq O_{h+1}$, $SV_{Os}[i] \neq SV_{Oh+1}[i]$ (lemma 4), this is a contradiction. So theorem 2 is correct.

If after time σ, there is at least one operation from other site is executed at s_i, or s_i generates at least one, then the saved local state is inconsistent with remote state at other sites. This feature is presented in theorem 3. Before proving theorem 3, we address five properties of history buffer as follows:

Property 2. If the generation time of O at s_i is earlier than t, then $e_i(O)$ is in the history buffer HB_i^t, thus, $\forall O, 1 \leq i \leq n$: $gt_i(O) < t \rightarrow e_i(O) \in HB_i^t$.

Property 3. If the execution time of O ($s(O) \neq i$) at s_i is earlier than t, then $e_i(O)$ is in the history buffer HB_i^t, thus, $\forall O, 1 \leq i \leq n$: $at_i(O) < t \rightarrow e_i(O) \in HB_i^t$.

Property 4. If the generation time of O at s_i is later than t, then $e_i(O)$ is not in the history buffer HB_i^t, thus, $\forall O, 1 \leq i \leq n$: $gt_i(O) > t \rightarrow e_i(O) \notin HB_i^t$.

Property 5. If the execution time of O ($s(O) \neq i$) at s_i is later than t, then $e_i(O)$ is not in the history buffer HB_i^t, thus, $\forall O, 1 \leq i \leq n$: $at_i(O) > t \rightarrow e_i(O) \notin HB_i^t$.

Property 6. Let θ be the time when s_i has failed, σ be the time when s_i generates the latest checkpoint and γ be the time when s_i begins its recovery procedure. For s_i, history buffer at time γ is the same as that at time δ, thus, $HB_i^\gamma = HB_i^\sigma$.

Theorem 3. If after time σ, there is at least one operation from other site is executed at s_i, or s_i generates at least one operation, HB_i^γ does not equal to HB_i^θ. In other words, $\exists O$: $(\sigma < gt_i(O) < \theta \lor \sigma < at_i(O) < \theta) \rightarrow HB_i^\gamma \neq HB_i^\theta$.

Proof. Firstly, suppose O is the operation that satisfies $\sigma < gt_i(O) < \theta$, we obtain that $e_i(O) \in HB_i^\theta$ (Property 2), and $e_i(O) \notin HB_i^\sigma$ (Property 4). So, we prove that $HB_i^\theta \neq HB_i^\sigma$. Since $HB_i^\gamma = HB_i^\sigma$ (Property 6), $HB_i^\gamma \neq HB_i^\theta$ is proved. Secondly, we suppose O is the operation that satisfies $\sigma < at_i(O) < \theta$, and $s(O) = j \neq i$, we have $e_i(O) \in HB_i^\theta$ (Property 2), and $e_i(O) \notin HB_i^\sigma$ (Property 5). So, we obtained $HB_i^\theta \neq HB_i^\sigma$. Because $HB_i^\gamma = HB_i^\sigma$ (Property 6), $HB_i^\gamma \neq HB_i^\theta$ is also proved. So Theorem 3 is correct.

Theorem 3 suggests that for all O, which satisfy $\sigma < gt_i(O) < \theta$ or $\sigma < at_i(O) < \theta$, are missing in HB_i^γ. If these operations are not reflected in HB_i^t, where $t > \gamma$, s_i is inconsistent with other sites. Therefore, consistency maintenance must be devised for the recovery procedure.

Assume that $y_i^{j, \sigma} = e_i(O_k)$, we found that operations O generated at s_j, $(1 \leq j \leq n, j \neq i)$, where $gt_j(O_k) < gt_j(O) < at_j(r)$, are also missing in HB_i^γ. The purpose of the recovery algorithm is to find out all the lost operations in s_i and the effect of their execution is remained unchanged. Hence, we introduce the consistency of the recovery as

follows:

Definition 9. Let σ, θ and γ be the latest checkpoint time, crash time and recovery time at s_i, the recovery is consistent: iff,

$\exists t > \gamma$: $\forall O$: $(\sigma < gt_i(O) < \theta \lor gt_j(O_k) < gt_j(O) < at_j(r)) \rightarrow e_i(O) \in HB_i^t$, where $y_i^{j, \sigma} = e_i(O_k)$.

Let s_j be the sites that receives the recovery message r from site s_i, $(i \neq j)$, s_j responds the message r at time t. The algorithm is given below.

Algorithm 3. the algorithm in s_j to respond the message r

Get $O_a \leftarrow y_i^{i, \sigma}$ and $O_b \leftarrow y_i^{j, \sigma}$ from the recovery message;
    $k \leftarrow 1$; $b_i \leftarrow$ false; $b_j \leftarrow$ false;
    if $y_i^{i, \sigma} = \varphi$ then $b_i \leftarrow$ ture;
    if $y_i^{j, \sigma} = \varphi$ then $b_j \leftarrow$ ture;
    while $k \leq |HB_j^t|$ do
        $O \leftarrow HB_j^t[k]$;
        if $(b_i = $ false$)$ then if $SV_O = SV_{Oa}$ then $b_i \leftarrow$ true;
        else if $s(O) = i$ then send $O' \leftarrow GORT(O)$ to s_i;
      if $(b_j = $ false$)$ then if $SV_O = SV_{Ob}$ then $b_j \leftarrow$ true;
      else if $s(O) = j$ then send $O' \leftarrow GORT(O)$ to s_i;
      $k \leftarrow k + 1$;
    end while;
end algorithm 3.

Note that the operations, which need to be sent back to the rejoin site again, are sent in the original form when they are generated. In fact, the operations in the history buffer are in the execution form instead of the original form, so we devise algorithm GORT (algorithm 4) to obtain the original form of an operation in history buffer.

Let HB_i^t be the history buffer of s_i at time t, $HB_i^t = [e_i(O_1), e_i(O_2), ..., e_i(O_m)]$, and $e_i(O_j)$ is an operation in HB_i^t.

In case that $\forall 1 \leq k \leq j\text{-}1$: $e_i(O_k) \rightarrow e_i(O_j)$, than original form of O_j is the same as its execution form, thus, $O_j = e_i(O_j)$.

Let $e_i(O_a)$ be the oldest operation that is independent of $e_i(O_j)$. In the simple case that $\forall 1 \leq k \leq a\text{-}1$: $e_i(O_k) \rightarrow e_i(O_j)$, and $\forall a \leq k \leq j\text{-}1$: $e_i(O_a) \| e_i(O_j)$, then we can directly obtain O_j by applying the list of exclusion transformation function (LET) [21], thus, $O_j = LET(e_i(O_j), HB_i^t[a, j\text{-}1]^{-1})$.

The complicated case is that there is a mixture of independent and dependent operations in the range of $HB_i^t[a, j\text{-}1]$. Let $EOL = [EO_{b1}, EO_{b2}, ..., EO_{br}]$ be the list of operations in the range of $HB_i^t[a+1, j\text{-}1]$, which are causally preceding $e_i(O_j)$. $EOL' = [EO'_{b1}, EO'_{b2}, ..., EO'_{br}]$, EO'_{bi} is the original form of operation EO_{bi}.

For the first operation in EOL, $EO'_{b1} = LET(EO_{b1}, HB_i^t[a, b1\text{-}1]^{-1})$. For the second operation in EOL, O_{b1} is determined by two step as follows, in which IT is the inclusion transformation function. IT is proposed in [21].
(1) $TO = LET(EO_{b2}, HB_i^t[a, b_{2\text{-}1}]^{-1})$;
(2) $EO'_{b2} = IT(TO, EO'_{b1})$.

Generally, for the ith operation in EOL, $(2 \le i \le r)$, the following two step is applied to obtain the corresponding form of operation in EOL,

(1) $TO = LET(EO_{bi}, HB_i^t[a, b_{i-1}]^{-1})$;

(2) $EO'_{bi} = IT(TO, [EO'_{b1}, EO'_{b2},..., EO'_{bi-1}]$.

If the operation list EOL′ is obtained, O_j can be easily get by the following two steps,

(1) $TO = LET(EO_{bi}, HB_i^t[a, j-1]^{-1})$;

(2) $O_j = IT(TO, EOL')$.

Algorithm 4. The Generic Operation Revise Transform algorithm, (GORT).

Given the history buffer of s_i at time t $HB_i^t = [e_i(O_1), e_i(O_2), ..., e_i(O_k)]$, and an operation $e_i(O_j)$ in HB_i^t, the original form of O_j is obtained as follows,

Scan HB_i^t from left to right to find the oldest operation $HB_i^t[a]$ that independent to $e_i(O_j)$;

if no such operation is found then return $O_j \leftarrow e_i(O_j)$;

Scan $HB_i^t[a, j-1]$ to find all operations that are causally preceding $e_i(O_j)$.

if no such operation is found

then return $O_j \leftarrow LET(e_i(O_j), HB_i^t[a, j-1]^{-1})$;

$EO'_{b1} \leftarrow LET(EO_{b1}, HB_i^t[a, b1-1]^{-1})$;

for $(2 \le i \le r)$ do

　$TO \leftarrow LET(EO_{bi}, HB_i^t[a, b_{i-1}]^{-1})$;

　　$EO'_{bi} \leftarrow IT(TO, [EO'_{b1}, EO'_{b2},..., EO'_{bi-1}])$;

　end for;

$TO \leftarrow LET(EO_{bi}, HB_i^t[a, j-1]^{-1})$;

return $O_j \leftarrow IT(TO, EOL')$;

end algorithm 4.

After each site s_j executes the algorithm 3, all lost operations in s_i will be executed again at s_i, and the effect of their execution is remained unchanged. This can be proved in theorem 4.

Assumption 1. There is at least one site s_j that, before time $at_j(r)$, has executed all operations generated at the failed s_i between time σ and θ, thus, $\exists 1 \le j \le n, j \ne i, t < at_j(r): \forall O: \sigma < gt_i(O) < \theta \rightarrow e_j(O) \in HB_j^t$.

Assumption 1 is very essential, for if no site executed all lost operations when recovery message arrives, then some lost operations will never be executed at s_i again. Thus, the consistency of the recovery can not be guaranteed.

Theorem 4. Our recovery algorithm generates a consistent recovery.

Proof. Assume that $y_i^{j, \sigma} = e_i(O_k)$. For site s_j ($1 \le j \le n$, and $j \ne i$), $y_j^{j, \delta} = e_j(LO_j)$ is the latest operation, where $\delta = at_j(r)$ is the arrival time of recovery message r from s_i to s_j. At time $t_j = at_i(LO_j)$, $e_i(LO_j) \in HB_i^{tj}$ (def. 5). Since recovery algorithm resends operations, which satisfy $s(O) = j$ and $O \rightarrow y_i^{j, \delta}$, to s_i; $y_i^{j, \sigma} \rightarrow y_j^{j, \delta}$, hence, $y_j^{j,\delta}$ is send to s_i again. Because $\forall e_j(O) \in HB_j^\delta: s(O) = j \rightarrow (O \rightarrow y_j^{j, \delta})$ (def. 8), we prove that at time t_j, $\forall O: gt_j(O_k) < gt_j(O) < at_j(r) \rightarrow e_i(O) \in HB_i^{tj}$, (property of causality preservation). Thus, we obtain $t_\alpha =$

$$MAX_{1 \le j \le n, j \ne i}(t_j) = MAX_{1 \le j \le n, j \ne i}(at_i(LO_j))$$

At time t_α, we have $\forall O, 1 \le j \le n, j \ne i: gt_j(O_k) < gt_j(O) < at_j(r) \rightarrow e_i(O) \in HB_i^{t\alpha}$. (1)

According to assumption 1, let s_k be the site that has executed all operations generated at s_i between σ and θ, thus, $\exists t < \delta: \forall O: \sigma < gt_i(O) < \theta \rightarrow e_k(O) \in HB_k^t$. So, we obtain $\forall O: \sigma < gt_i(O) < \theta \rightarrow e_k(O) \in HB_k^\delta$ (2)

Let δ be the arrival time of recovery message from s_i to s_k, $\delta = at_k(r)$, and $y_k^{i, \delta} = e_k(LO_k')$ is the latest operation from s_i in HB_k^δ. As described in our algorithm, these operations are sent back to s_i again, we have $\exists t_\beta = at_i(LO_k') > \delta: e_i(LO_k') \in HB_i^{t\beta}$. Since $\forall e_k(O) \in HB_k^\delta: s(O) = i \rightarrow (O \rightarrow LO_k')$, we prove that at time t_β, $\forall e_k(O) \in HB_k^\delta: s(O) = i \rightarrow e_i(O) \in HB_i^{t\beta}$ (3) (based on causality property).

Based on (2) and (3), we prove that at time t_β, $\forall O: \sigma < gt_i(O) < \theta \rightarrow e_i(O) \in HB_i^{t\beta}$ (4). According to (1) and (4), we have $\exists t = MAX(t_\alpha, t_\beta) > \gamma: \forall O: (s(O) = i \land \sigma < gt_i(O) < \theta) \lor (s(O) = j \ne i \land gt_j(O_k) < gt_j(O) < at_j(r)) \rightarrow e_i(O) \in HB_t^i$, where $y_i^{j, \sigma} = e_i(O_k)$. Thus, the recovery is consistent.

4. Performance Analysis

This section evaluates the performance of our new approach of recovery support for collaborative editing systems. We assume that when a site leaves the collaborative editing system successfully, it had created m checkpoints. The expected interval time between a site join and leave the system reflects the performance of the system. Under the same workload, the shorter the interval time, the better performance the system. We investigate the factors that make significant impact on this interval time.
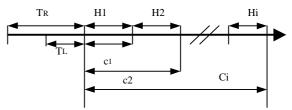

Figure 2. Definition for ci, Hi, TL and TR

As displayed in figure 2, P_i $(2 \le i \le m)$ represents the execution time on site, it is the nominal measured in CPU cycles between (i-1)th and ith checkpoints, P_1 denotes the interval between the beginning of the site 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 $(1 \le i \le m)$ be the execution time from the

beginning of the site to the ith checkpoint in presence of the site 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 site between join and leave the collaborative editing system is $C_m = E(c_m)$.

The site and the Internet link failures can be recovered by either loading LDS or remote document state. Let p and q be the probability of recovering the site by using our new LDS approach and the traditional RDS approach, respectively, it is clear that $p + q = 1$. Let T_L and T_R denote time overhead for loading LDS and RDS, respectively. $f_i(t)$ ($i \in [2, m]$) denotes the probability of a site/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,

$$c_1 = \begin{cases} P_1 & \text{with probability } 1 - f_1(P_1) \\ P_1 + T_L + c_1 & \text{with probability } p \times f_1(P_1) \\ P_1 + T_R + c_1 & \text{with probability } q \times f_1(P_1) \end{cases} \quad (2)$$

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

$$c_i = c_{i-1} + H_i + T_C \quad (3)$$
$$H_1 = c_1 \quad (4)$$

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

C_i is derived from equations (5) as follows, where $2 \leq i \leq m$,

$$C_i = \frac{C_{i-1} + P_i + (pT_L + qT_R) f_i(P_i)}{1 - f_i(P_i)} \quad (6)$$

C_m represents the expected interval time of the site between join and leave the system, it is obtained by applying the above equation m-1 times,

$$C_m = \sum_{j=1}^{m} \prod_{i=j}^{m} \{[P_j + (pT_L + qT_R) f_j(P_j)] / [1 - f_i(P_i)]\} \quad (7)$$

The value of C_m reflects the performance of the system. Hence, in order to enhance the performance, C_m can be minimized by determining the proper checkpointing frequency. The value of m that minimizes the equation (7) is an optimal one.

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

$$C^L(P,k) = (P+T_L)p_1^S + 2(P+T_L)p_1^U p_2^S + \ldots + k(P+T_L)$$
$$\prod_{i=1}^{k-1} p_i^U p_k^S + \left[k(P + T_L) + \frac{P + T_R}{1 - f_1(P)} \right] \prod_{i=1}^{k} p_i^U$$
$$= \sum_{j=1}^{k-1} [j(P+T_L)] \prod_{i=1}^{j-1} p_i^U p_j^S + k(P+T_L) \prod_{i=1}^{k-1} p_i^U$$

$$+ \left[\frac{P + T_R}{1 - f_1(P)} \right] \prod_{i=1}^{k-1} p_i^U \quad (8)$$

p_i^S and p_i^U are not known until (i-1)th unsuccessful LDS recovery occurs. We derive the approximate probability for p_i^S and p_i^U. It is clear that with the increase number of unsuccessful LDS recoveries, the probability of permanent rises, thus,

$$p_1^U < p_2^U < \ldots < p_k^S \text{ and } p_1^S > p_2^S > \ldots > p_k^S \quad (9)$$

We assume that $p_i^S/p_{i-1}^S = w_i < 1$, and for the simplicity, it is assumed that $w_1 = w_2 = \ldots = w_k = w$, and $p_1^S = p$. Equation 9 is derived from equation 8 as follows,

$$C^L(P,k) = \sum_{j=1}^{k-1} [j(P+T_L)] \prod_{i=1}^{j-1} (1 - pw^{i-1})$$
$$+ k(P + T_L) \prod_{i=1}^{k-1} (1 - pw^{i-1}) +$$
$$[(P + T_R)/(1 - f_1(P))] \prod_{i=1}^{k} (1 - pw^{i-1}) \quad (10)$$

The time overhead of LDS recovery is determined by P and the arrival rate of operations λ. Suppose the operation arrival rate is constant, hence, with the increase of P, the probability of successful LDS recovery decreased, and the time overhead of the unsuccessful LDS also increases. On the other hand, the time overhead of RDS recovery is decided by the data volume associated with the context of the document. For the simplicity, we assume that the cost of the RDS recovery remains constant, and it is modelled as follows,

$$C^R(P) = \frac{P + T_R}{1 - f_1(P)} \quad (11)$$

LDS recovery is an efficient method to recover the temporary failures in site and links. It continues working until the permanent failure occurs (checkpoint stored on local storage is missing) or the time overhead of LDS recovery is larger than RDS recovery. Thus, given value P, $C^L(P,k)$ can be determined by k, which must satisfies $C^L(P,k) < C^R(P)$.

The table 1 describes the relation between k and $C^L(P,k)$. P is set to 100, 200 and 300, respectively. C^L first decreases with the increase of k, and when k = 12, C^L is the minimized. After k = 12, CL rises with the increase of k. In this case, 12 is the optimal value for k.

Table 1. $T_L = 20$, $T_R = 40$, $w = 0.8$, $p = 0.8$, $f_1(P) = 0.1$

k	2	4	6	8	10	12
P=100	179.24	177.61	170.46	167.69	166.68	166.14
P=200	327.2	325.22	312.32	307.31	305.47	305.00
P=300	475.2	473.85	454.18	446.92	444.28	443.59
K	14	16	18	30	50	100
P=100	166.53	166.86	167.36	171.56	179.56	199.77
P=200	305.22	305.85	306.76	314.56	329.14	366.18
P=300	443.92	444.85	446.15	457.36	478.71	532.60

To study the impact of the probability of the first

successful LDS recovery on $C^L(P,k)$, we fixed T_L, T_R, w, and $f_1(P)$, and increased k from 10 to 30 with increments of 10. Table. 2 shows the execution time of the site in the presence of up to k LDS recovery as a function of p. The higher the probability p is, the less execution time of the site in the presence of up to k LDS recovery is. It suggests that a higher probability of the first successful LDS recovery result in a better performance.

Table 2. P=100, T_L=20, T_R = 40, w = 0.8, f_1(P) = 0.1

p	0.65	0.70	0.75	0.8	0.85	0.9
K=10	208.48	192.86	179.17	166.68	154.84	143.26
K=20	217.40	197.87	181.80	167.94	155.37	143.41
K=30	234.33	208.49	188.18	171.56	157.25	144.29

Table 3 illustrates the relation between w and $C^L(P,k)$. T_L, T_R, p, and $f_1(P)$ are fixed, and k is set to 10, 20 and 30, respectively. Like the effect of p on C^L, as the value of w rises, the execution time of the failed site in the presence of up to k LDS recovery decreases. This is because with the increase of value w, the probability of ith unsuccessful LDS recovery decreases, and as p_i^U drops, C^L decreases. This suggests that if we could increase the probability of the successful LDS recovery, the performance of the system is enhanced.

Table 3. P = 100, T_L = 20, T_R = 40, p = 0.8, f_1(P) = 0.1

W	0.65	0.70	0.75	0.8	0.85	0.9
K=10	195.14	181.91	172.25	166.68	164.94	165.98
K=20	231.97	202.70	180.73	167.94	164.01	165.48
K=30	270.74	226.14	192.10	171.56	164.40	165.44

5. Related Work

Collaborative editing has been studied deeply. Collaborative editing systems can be classified into two categories: Real-time and non real-time [15]. 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][26]. In real-time collaborative editing systems, multiple users are allowed to concurrently edit any part of the document simultaneously. 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 user intention preservation [8], consistency maintenance [20] [21], group undo [18], and group awareness [6][24][25], 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 sites. Paper [16] provides the practical fault-tolerant services for collaborative system. Since these systems are designed for general purpose, they do not study the concurrency control issues.

Paper [7] presents the requirements for the collaborative editor, and proposes a model that has fault-tolerant feature. This technique is also discussed in paper [1]. However, 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 fail 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.

6. Conclusion

An efficient recovery algorithm is presented to make the real-time collaborative systems more reliable. In our new approach, each site maintains a LDS, which is generated periodically. If a failure occurs in the site or links, the site is able to rejoin the collaborative editing systems by loading the LDS instead of obtaining the state from other sites that may result in a noticeable delay

In our model of performance evaluation, the time overhead of LDS recovery was modelled as a constant. In fact, it was determined by the frequency of taking a checkpoint and the arrival rate of operations. It is not known how these two factors affect the time overhead of LDS recovery. Implications of big differences. Further work needed.

An assumption implied in the model that each site maintains a history buffer with an infinite storage capacity is not practical. However, the purpose of the history buffer is to support the undo/transform-do/transform-redo scheme and the recovery, an operation need not to be kept into the history buffer if is it is no longer involved in these two procedures. Therefor, our continuous work focuses on devising the technique for garbage collection, which overcomes this storage problem.

Reference

[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 Coodination 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.121-1301999.

[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 intergrating, transformation-oriented approach to concurrency control and undo in goup 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, Y. Yang, Y. Zhang, and D. Chen. Distributed Concurrency Control in Real-time Cooperative Editing Systems. *In Proceedings of the Asian Computing Science Conference*, Lecture Notes in Computer Science, Vol. 1179. Springer-Verlag, 84-95, Dec. 1996.

[21] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieiving 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.

[25] W. Prinz. NESSIE: An awareness environment for cooperative settings. *In Proceedings of the Sixth European conference on Computer Supported Cooperative Work, Kluwer Academic Publishers,* pp. 391-410, 1999.

[26] E.J. Whitehead, Jr. and Y.Y. Goland. WebDAV: A network protocol for remote collaborative authoring on the Web. *In Proceedings of the Sixth European conference on Computer Supported Cooperative Work, Kluwer Academic Publishers,* pp. 291-310, 1999.

[27] J. Trevor, T. Koch, and G. Woetzel. MetaWeb: Bringing synchronous groupware to the World Wide Web. *In Proceedings of the fifth European conference on Computer Supported Cooperative Work, Kluwer Academic Publishers,* pp. 65-80, 1997.