Rollback-Recovery Protocol Guarantying MR Session Guarantee in Distributed Systems with Mobile Clients[⋆]

Jerzy Brzeziński, Anna Kobusińska, and Michał Szychowiak

Institute of Computing Science
Poznań University of Technology, Poland
{Jerzy.Brzezinski,Anna.Kobusinska,Michal.Szychowiak}@cs.put.poznan.pl

Abstract. This paper presents rVsMR rollback-recovery protocol for distributed mobile systems, guarantying Monotonic Reads consistency model, even in case of server's failures. The proposed protocol employs known rollback-recovery techniques, however, while applying them, the semantics of session guarantees is taken into account. Consequently, rVsMR protocol is optimized with respect to session guarantees requirements. The paper includes the proof of safety property of the presented protocol.

Keywords: rollback-recovery, safety, mobile systems, Monotonic Reads session guarantee

1 Introduction

Applications in mobile domain usually tend to be structured as client-server interactions. In such applications, clients accessing the data are not bound to particular servers, but they can switch from one server to another. This switching adds a new dimension of complexity to the problem of consistency and makes the management of data consistency from client's perspective very attractive. Therefore, in [TDP$^+$94] a new class of consistency models, called *session guarantees* (or *client-centric* consistency models), has been proposed to define properties of the system, observed from client's point of view. Client-centric consistency models define four session guarantees: *Read Your Writes* (RYW), *Monotonic Writes* (MW), *Monotonic Reads* (MR) and *Writes Follow Reads* (WFR). RYW expresses the user expectation not to miss his own modifications performed in the past, MW ensures that order of writes issued by a single client is preserved, MR ensures that the client's observations of the

⋆ This work was supported in part by the State Committee for Scientific Research (KBN), Poland, under grant KBN 3 T11C 073 28 data storage are monotonic and finally, WFR keeps the track of causal dependencies resulting from operations issued by a client.

In this paper we focus our attention on MR session guarantee. Below we give a couple of examples that demonstrate the usefulness of MR. First, let us imagine a mailbox of a traveling user, who opens the mailbox at one location, reads emails, and afterwards opens the same mailbox at different location. The user should see at least all the messages he has read previously, which is impossible without MR. Further, imagine that user's appointment calendar is stored on-line in replicated database, and can be updated by both: the user and automatic meeting scheduler. The calendar program periodically refreshes its display by reading appointments from the database. The recently added (or deleted) meetings can not appear to come and go, which is ensured, when copies of the database held by servers are consistent with respect to MR [TDP+94]. Finally, consider a Web page replicated at two different stores S_1 and S_2. If a client first reads the page from S_1 and later again from S_2, then the second copy should be the same, or newer as the one read from S_1.

MR session guarantee is provided by appropriate consistency protocols [TDP+94,BSW05]. In order to construct effective solutions, adjusted to real application requirements, these protocols should provide MR also in situations, when servers holding replicated data brake down. Unfortunately, as far as we know, none of the proposed consistency protocols preserving session guarantees, considers such a possibility; they generally assume non-faulty environments. Such assumption might be considered not plausible and too strong for certain mobile distributed systems, where in practice failures do happen. Therefore, this paper addresses a problem of providing MR session guarantee in case of server's failures.

We introduce the rollback-recovery protocol rVsMR for distributed mobile systems, which combines fault–tolerant techniques: logging and checkpointing with coherence operations of a formerly proposed VsSG consistency protocol [BSW05]. As a result, the rVsMR protocol offers the ability to overcome the servers' failures, at the same time preserving MR session guarantee. Because of client's orientation, in rVsMR protocol runtime faults are corrected with any intervention from the user. The main contribution of this paper is a presentation of rollback-recovery protocol rVsMR of MR session guarantee and formal proof of its safety.

2 Related work

Session guarantees have been introduced in the context of Bayou replicated storage system [TDP$^+$94] to allow mobile clients to implicitly define sets of writes that must be performed by servers. Since in Bayou each server's state is maintained in the database, adding a persistent and crash resisting log is enough to provide fault–tolerance in case of server's failure. CASCADE — a caching service for distributed CORBA objects [CDFV00], is another system using consistency conditions based on session guarantees. In CASCADE it is assumed that processes do not crash during the execution and all communication links are eventually operational. The Globe system [KKvST98] follows the approach similar to CASCADE, by providing a flexible framework for associating various replication coherence models with distributed objects. Among the coherence models supported by Globe are also client-based models, although they are combined with object-based consistency models in a single framework. Finally, Pastis — a highly scable, multi-user, peer-to-peer file system [PBS05] implements a consistency model based on RYW session guarantee. In Pastis it is assumed that at least one replica is not faulty and all users allowed to write to a given file trust one another regarding the update of that file.

3 System model, basic definitions and notations

Throughout this paper, a replicated distributed storage system is considered. The system consists of a number of unreliable *servers* holding a full copy of a *shared objects* and *clients* running applications that access these objects. Clients are mobile, i.e. they can switch from one server to another during application execution. To access the shared object, clients select a single server and send a direct request to this server. Operations are issued by clients sequentially, i.e. a new operation may be issued after the results of the previous one have been obtained. In this paper we focus on failures of servers, and assume the *crash-recovery* failure model, i.e. servers may crash and recover after crashing a finite number of times [GR04]. Servers can fail at arbitrary moments and we require any such failure to be eventually detected, for example by failure detectors [SDS99].

The storage replicated by servers does not imply any particular data model or organization. Operations performed on shared objects are basically divided into *reads* and *writes*. The server, which first obtains the write from a client, is responsible for assigning it a globally unique identifier. Clients can concurrently submit conflicting writes at different servers, e.g. writes that modify the overlapping parts of data storage. Operations on shared objects issued by client C_i are ordered by a relation $\overset{C_i}{\rightarrow}$ called *client issue order.* A server S_j performs operations in an order represented by a relation $\overset{S_j}{\rightarrowtail}$. Operations on objects are denoted by w, r or o, depending on the operation type (write, read or these whose type is irrelevant). Every server maintains the set CR_{S_j} of indexes of clients from which it has directly received write requests and table RW_{S_j}, where the number of writes performed by S_j before read from C_i was obtained, is kept in position i. Relevant writes $RW(r)$ of a read operation r is a set of writes that has influenced the current state of objects observed by the read r. Formally, MW session guarantee is defined as follows [BSW05]:

Definition 1. *Monotonic Reads (MR) session guarantee is a property meaning that:*

$$\forall C_i \, \forall S_j \left[r_1 \overset{C_i}{\rightarrow} r_2|_{S_j} \implies \forall w_k \in RW(r_1) : w_k \overset{S_j}{\rightarrowtail} r_2 \right]$$

In the paper, it is assumed, that data consistency is managed by the VsSG *consistency protocol* [BSW05]. The formerly proposed protocol VsSG [BSW05] uses a concept of server-based version vectors for efficient representation of sets of writes required by clients. Server-based version vectors have the following form: $V_{s_j} = \left[v_1 \, v_2 \, ... \, v_{N_S} \right]$, where N_S is a total number of servers in the system and single position v_i is the number of writes performed by server S_j. Every write w in the VsSG protocol is labeled with a *vector timestamp*, denoted by $T(w)$ $(T : \mathcal{O} \mapsto V\ )$ and set to the current value of the vector clock V_{S_j} of server S_j performing w for the first time. During writes, performed by server S_j, its version vector V_{S_j} is incremented in position j and a timestamped operation is recorded in history H_{S_j}. $\mathcal{O}_{S_j}$ is a set of all writes performed by the server in the past. The writes that belong to $\mathcal{O}_{S_j}$ come from direct requests received by S_j from clients or are incorporated from other servers during the synchronization procedure. The VsSG protocol eventually propagates all writes to all servers. At the client's side, vector R_{C_i} representing writes relevant to reads issued by the client C_i is maintained. The linearly ordered set $\left(\mathcal{O}_{S_j}, \overset{S_j}{\rightarrowtail} \right)$ of past writes is denoted by H_{S_j} and called *history* [BSW05]. During synchronization of servers, their histories are *concatenated.* The concatenation of histories H_{S_j} and H_{S_k}, denoted by $H_{S_j} \oplus H_{S_k}$, consists in adding new operations from H_{S_k} at the end of H_{S_j}, preserving at the same time the appropriate relations [BSW05].

Below, we propose formal definitions of fault-tolerance mechanisms used by the rVsMR protocol:

Definition 2. *A log Log_{S_j} is a set of triples:*

$$\left\{ \langle i_1, o_1, T(o_1) \rangle \; \langle i_2, o_2, T(o_2) \rangle \; ... \; \langle i_n, o_n, T(o_n) \rangle \right\},$$

where i_n represents the identifier of the client issuing a write operation $o_n \in \mathcal{O}_{S_j}$ and $T(o_n)$ is timestamp of o_n.

Definition 3. *Checkpoint $Ckpt_{S_j}$ is a couple $\left\langle V_{S_j}, H_{S_j} \right\rangle$, of version vector V_{S_j} and history H_{S_j} maintained by server S_j at the time t, where t is a moment of taking a checkpoint.*

In this paper we assume, that log and checkpoint are saved by the server in a *stable storage*, able to survive all failures [EEL$^+$02]. Additionally, we assume that the newly taken checkpoint replaces the previous one, so just one checkpoint for each server is kept in the stable storage.

4 The rVsMR protocol

For every client C_i that requires MR session guarantee when executing read r, results of all writes, which have influenced the read issued by a client before r cannot be lost. Unfortunately, at the moment of performing the operation, the server does not possess the knowledge, whether in the future the client will be interested in reading results of its writes or not. So, to preserve MR, the recovery protocol should ensure that outcomes of all writes performed by the server are not lost in the case of its failure.

In the proposed rVsMR protocol, we introduce a novel optimization that reduces the number of saved operations: we propose that every server S_j saves only operations obtained directly from clients. Although only some of operations performed by S_j are saved, we prove that MR is fulfilled in case of S_j failure.

The server that obtains the write request directly from client C_i, logs the request to stable storage (Figure 1, l. 13), and only afterwards performs it (l. 14). The moment of taking a checkpoint is determined by obtaining a read request r_2, which follows another read r_1 issued by the same client. The server, which obtains operation r_2 from a client, checks first, whether such a read can be performed (by comparing the values of vectors V_{S_j} and W - l. 6). When performing read r_2 is possible, the server checks if it has already performed, since the latest checkpoint, any write operation that influenced the state of objects observed by the read

```
Upon sending a request $\langle o \rangle$
to server $S_j$ at client $C_i$

$W \leftarrow \mathbf{0}$
if (not iswrite($o$)) then
    $W \leftarrow \max(W, R_{C_i})$
end if
send $\langle o, W \rangle$ to $S_j$

Upon receiving a request $\langle o, W \rangle$
from client $C_i$ at server $S_j$

while $\left(V_{S_j} \not\geq W\right)$ do
    wait()
end while
if iswrite($o$) then
    $CW_{S_j} \leftarrow CW_{S_j} \cup i$
    $V_{S_j}[j] \leftarrow V_{S_j}[j] + 1$
    timestamp $o$ with $V_{S_j}$
    $Log_{S_j} \leftarrow Log_{S_j} \cup \langle i, o, T(o) \rangle$
    perform $o$ and store results in $res$
    $H_{S_j} \leftarrow H_{S_j} \oplus \{o\}$
    $nWrites \leftarrow nWrites + 1$
end if
if not iswrite($o$) then
    if $i \in CR_{S_j}$ then
        $secondRead \leftarrow TRUE$
    else
        $CR_{S_j} \leftarrow CR_{S_j} \cup i$
        $RW_{S_j}[i] \leftarrow nWrites$
    end if
    if $(RW_{S_j}[i] > 0)$ and $secondRead$ then
        $Ckpt_{S_j} \leftarrow \langle V_{S_j}, H_{S_j} \rangle$
        $Log_{S_j} \leftarrow \emptyset$
        $CR_{S_j} \leftarrow \emptyset$
        $secondRead \leftarrow FALSE$
        $nWrites \leftarrow 0$
        $RW_{S_j} \leftarrow \mathbf{0}$
    end if
    perform $o$ and store results in $res$
end if
send $\left\langle o, res, V_{S_j} \right\rangle$ to $C_i$

Upon receiving a reply $\langle o, res, W \rangle$
from server $S_j$ at client $C_i$

if iswrite($o$) then
    $R_{C_i} \leftarrow \max(R_{C_i}, W)$
end if
deliver $\langle res \rangle$

Every $\Delta t$ at server $S_j$
foreach $S_k \neq S_j$ do
    send $\left\langle S_j, H_{S_j} \right\rangle$ to $S_k$
end for

Upon receiving an update $\langle S_k, H \rangle$
at server $S_j$

foreach $w_i \in H$ do
    if $V_{S_j} \not\geq T(w_i)$ then
        perform $w_i$
        $V_{S_j} \leftarrow \max\left(V_{S_j}, T(w_i)\right)$
        $H_{S_j} \leftarrow H_{S_j} \oplus \{w_i\}$
    end if
end for
signal()

On rollback-recovery
$\langle V_{S_j}, H_{S_j} \rangle \leftarrow Ckpt_{S_j}$
$CR_{S_j} \leftarrow \emptyset$
$secondRead \leftarrow FALSE$
$nWrites \leftarrow 0$
$RW_{S_j} \leftarrow \mathbf{0}$
$Log'_{S_j} \leftarrow Log_{S_j}$
$vrecover \leftarrow \mathbf{0}$
while $\{\, o'_j : T(o'_j) > vrecover \} \neq \emptyset$ do
    choose $\langle i', o'_i, T(o'_i) \rangle$ with minimal $T(o'_j)$ from $Log'_{S_j}$ where $T(o'_j) > V_{S_j}$
    $V_{S_j}[j] \leftarrow V_{S_j}[j] + 1$
    perform $o'_j$
    $H_{S_j} \leftarrow H_{S_j} \oplus \{o'_j\}$
    $CW_{S_j} \leftarrow CW_{S_j} \cup i'$
    $vrecover \leftarrow T(o'_i)$
    $nWrites \leftarrow nWrites + 1$
end while
```

Fig. 1. Checkpointing and rollback-recovery rVsMR protocol

r_1 (l. 25). When at least one such write has been performed, the server checkpoints its state (l. 26), performs the read operation (l. 33) and sends a reply to the client (l. 35). Otherwise, the new checkpoint need not be taken. After the checkpoint is taken, server logs are cleared (l. 27). Saving the state of server earlier would be unnecessary, as when write request is not followed by a read one, it does not violate MR. Essential is the fact, that first the checkpoint is taken, and only afterwards the content of log Log_{S_j} is cleared. (l. 27). After the failure occurrence, the failed server restarts from the latest checkpoint (l. 51) and replays operations from the log (l. 58-65) according to their timestamps, from the earliest to the latest one. Writes received from other servers during update procedure, and missing from the local history of S_j, are performed, but not logged (l. 45-47). Thus, such writes are lost after the failure occurrence. However, those writes are saved in the log or in the checkpoint of servers, which received them directly from clients. Hence, lost writes will be eventually obtained again in consecutive synchronizations.

5 Safety of rVsMR protocol

Lemma 1. *Every write operation w issued by client C_i and performed by server S_j that received w directly from client C_i, is kept in checkpoint $Ckpt_{S_j}$ or in log Log_{S_j}.*

Proof. Let us consider write operation w issued by client C_i and obtained by server S_j.

1. From the algorithm, server S_j before performing the request w, saves it in the stable storage by adding it to log Log_{S_j} (l. 13). Because logging of w takes place before performing it (l. 14), then even in the case of failure operation w is not lost, but remains in the log.
2. Log Log_{S_j} is cleared after performing by S_j the second read request issued by the same client. However, according to the algorithm, read operations cause storing the information on writes by checkpointing the server's version vector V_{S_j} and history H_{S_j} in $Ckpt_{S_j}$ (l. 26). The checkpoint is taken before clearing log Log_{S_j} (l. 27). Therefore, the server failure, which occurs after clearing the log, does not affect safety of the algorithm because writes from the log are already stored in the checkpoint.

Lemma 2. *The rollback-recovery procedure recovers all write operations issued by clients and performed by server S_j that were logged in log Log_{S_j} in the moment of server S_j failure.*

Proof. Let us assume that server S_j fails. The rollback-recovery procedure recovers operations remembered in the log (l. 58), after recovering V_{S_j} and H_{S_j} from a checkpoint (l. 51). The recovered operation updates version vector V_{S_j} (l. 60), is performed by S_j (l. 61) and added to the server's S_j history H_{S_j}(l. 62).

Assume now, that failures occur during the rollback-recovery procedure. Due to such failures the results of operations that have already been recovered are lost again. However, since log Log_{S_j} is cleared only after the checkpoint is taken (line 27) and it is not modified during the rollback-recovery procedure (l. 56), the log's content is not changed. Hence, the recovery procedure can be started from the beginning without loss of any operation issued by clients and performed by server S_j after the moment of taking checkpoint.

Lemma 3. *Operations obtained and performed in the result of synchronization procedure and required by MR, are performed again after the failure of S_j, before processing a new read from a client.*

Proof. By contradiction, let us assume that server S_j has performed a new read operation r obtained from client C_i before performing again operation w, received during a former synchronization and lost because of S_j failure. According to VsSG protocol, before executing r the condition $V_{S_j} \geq R_{C_i}$ is fulfilled (l. 6) .

Further assume, that w issued by C_i before r, has been performed by server S_k. According to the protocol, after the reply from S_k is received by C_i, vector R_{C_i} is modified: $R_{C_i} \leftarrow \max(W, R_{C_i})$. This means that vector R_{C_i} is updated at least at position k: $R_{C_i}[k] \leftarrow k + 1$. (l. 37). Server S_j, during synchronization procedure with S_k, performs w and updates its version vector: $V_{S_j} \leftarrow \max\left(V_{S_j}, T(w)\right)$, which means that V_{S_j} has been modified at least in the position k (l. 46). However, if failure of S_j happens, the state of S_j is recovered accordingly to values stored in the checkpoint $Ckpt_{S_j}$ (l. 51) and in the log Log_{S_j} (l. 58-65). From the algorithm, while recovering operations from the log, the vector V_{S_j} is updated only at position j. Thus, if operation w_1 performed by S_j in the result of synchronization with server S_k is lost because of S_j failure, the value of $V_{S_j}[k]$ does not reflect the information on w. Hence, until the next update message is obtained, $V_{S_j}[k] < R_{C_i}[k]$, which contradicts the assumption.

Lemma 4. *The recovered server performs new read operation issued by a client only after all writes performed before the failure and required by MR are restored.*

Proof. By contradiction, let us assume that there is a write operation w performed by server S_j before the failure occurred, that has not been recovered yet, and that the server has performed a new read operation issued by client C_i. According to original VsSG protocol [BSW05], managing only consistency not reliability issues, for reliable server S_j that performs new read operation, the condition $V_{S_j} \geq R_{C_i}$ is fulfilled (l. 6-7).

Let us consider which actions are taken when a write operation is issued by client C_i and performed by server S_j. On the server side, the receipt of the write operation causes the update of vector V_{S_j} in the following way: $V_{s_j}[j] \leftarrow V_{S_j}[j] + 1$ and results in timestamping w with the unique identifier (l. 12). The server that has performed the write sends a reply containing the modified vector V_{S_j} to the client. At the client side, after the reply is received, vector R_{C_i} is modified: $R_{C_i} \leftarrow \max(W, R_{C_i})$ (l. 37). This means that vector R_{C_i} is updated at least at position j: $R_{C_i}[j] \leftarrow \max[j] + 1$. If there is a write operation w performed by server S_j before the failure that has not been recovered yet, then $V_{S_j}[j] < R_{C_i}[j]$, which follows from the ordering of recovered operations (l. 59). This is a contradiction with $V_{S_j} \geq R_{C_i}$. Hence, the new read operation cannot be performed until all previous writes are recovered.

Theorem 1. *MR session guarantee is preserved by rVsMR protocol for clients requesting it, even in the presence of server failures.*

Proof. It has been proven in [BSW05] that VsSG protocol preserves MR session guarantee, when none of servers fails. According to Lemma 1, every write operation performed by server S_j is saved in the checkpoint or in the log. After the server's failure, all operations from the checkpoint are recovered. Further, all operations performed before the failure occurred, but after the checkpoint was taken, are also recovered (according to Lemma 2). According to Lemma 4, all recovered write operations are applied before new reads are performed. Moreover, operations obtained by S_j during synchronization procedure and lost because of S_j failure, are also performed once again before new reads from C_i (from Lemma 3). Hence, for any client C_i and any server S_j, MR session guarantee is preserved.

Full versions of the theorems and proofs can be found in [BKS05].

6 Conclusions

Although our implementation of rollback-recovery protocol is based on the known techniques of operation logging and checkpointing of server's state, it is nevertheless unique in exploiting properties of Monotonic Reads session guarantee while applying these techniques. This results in checkpointing only the results of write operations, which are essential to provide MR. Furthermore, we have designed novel optimisations that reduce the number of saved operations. We believe that rVsMR protocol can be applied to other systems (Section 2), where it is required to maintain consistency for mobile clients.

Our future work encompasses the development of rollback-recovery protocols, which are integrated with other consistency protocols. Moreover, appropriate simulation experiments to quantitatively evaluate overhead of rVsMR protocol are being carried out.

References

[BKS05] J. Brzeziński, A. Kobusińska, and M. Szychowiak. Mechansim of rollback-recovery in mobile systems with mr. Technical Report RA-012/05, Institute of Computing Science, Poznań University of Technology, November 2005.

[BSW05] J. Brzeziński, C. Sobaniec, and D. Wawrzyniak. Safety of a server-based version vector protocol implementing session guarantees. In *Proc. of Int. Conf. on Computational Science (ICCS2005), LNCS 3516*, pages 423–430, Atlanta, USA, May 2005.

[CDFV00] G. Chockler, D. Dolev, R. Friedman, and R. Vitenberg. Implementing a caching service for distributed CORBA objects. In *Proc. of Middleware 2000: IFIP/ACM Int. Conf. on Distributed Systems Platforms*, pages 1–23, April 2000.

[EEL$^+$02] N. Elmootazbellah, Elnozahy, A. Lorenzo, Yi-Min Wang, and D.B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, September 2002.

[GR04] Rachid Guerraoui and Luis Rodrigues. *Introduction to distributed algorithms.* Springer-Verlag, 2004.

[KKvST98] Anne-Marie Kermarrec, Ihor Kuz, Maarten van Steen, and Andrew S. Tanenbaum. A framework for consistent, replicated Web objects. In *Proc. of the 18th Int. Conf. on Distributed Computing Systems (ICDCS)*, May 1998.

[PBS05] F. Picconi, J-M. Busca, and P. Sens. Pastis: a highly-scalable multi-user peer-to-peer file system. *EuroPar 2005*, pages 1173–1182, 2005.

[SDS99] N. Sergent, X. Défago, and A. Schiper. Failure detectors: Implementation issues and impact on consensus performance. Technical Report SSC/1999/019, École Polytechnique Fédérale de Lausanne, Switzerland, May 1999.

[TDP$^+$94] Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent W. Welch. Session guarantees for weakly consistent replicated data. In *Proc. of the Third Int. Conf. on Parallel and Distributed Information Systems (PDIS 94)*, pages 140–149, Austin, USA, September 1994. IEEE Computer Society.