SAFETY OF CHECKPOINTING AND ROLLBACK-RECOVERY PROTOCOL FOR MOBILE SYSTEMS WITH RYW SESSION GUARANTEE[*]

Jerzy Brzeziński, Anna Kobusińska, Jacek Kobusiński
Poznań University of Technology, Institute of Computing Science
ul. Piotrowo 2, 60-965 Poznań, Poland

Keywords: Rollback-recovery, mobile systems, Read Your Writes session guarantee.

Abstract: This paper presents rVsRYW checkpointing and rollback-recovery protocol, which preserves Read Your Writes session guarantee and includes the proof of safety property of the proposed protocol. In order to provide RYW guarantee required by mobile clients, despite failures of servers, in rVsRYW protocol semantics of consistency protocol operations is exploited and build in recovery mechanisms.

1 INTRODUCTION

Mobile environments, by enabling motion and location independence of clients, give the opportunity to provide new services and allow supplementary information access that may occur any time and any place. In such environments, clients accessing the data may not be 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, because after switching to another server, client's new operations should remain consistent with ones previously issued. Therefore, a new class of consistency models, called session guarantees, recommended for mobile environment, has been introduced (Terry et al., 1994). Session guarantees, also called client-centric consistency models, define required properties of the system observed from clients point of view. Four session guarantees have been defined: Read Your Writes (RYW), Monotonic Writes (MW), Monotonic Reads (MR) and Writes Follow Reads (WFR) and protocols implementing them have been introduced (Terry et al., 1994; Kobusińska et al., 2005; Sobaniec, 2005). However, according to our knowledge, none of proposed consistency protocols is resistant to servers' failures. Such assumption might be considered not plausible and too strong for certain mobile distributed systems. Therefore, in this paper checkpointing and rollback-recovery protocol rVsSG that preserves RYW session guarantee is introduced. rVsRYW protocol ensures that after the server failure and its recovery, RYW session guarantee is preserved. In the proposed protocol the semantics of operations of VsSG consistency protocol (Kobusińska et al., 2005; Sobaniec, 2005) is exploited and build in recovery mechanisms. As a result, rVsRYW protocol offers the ability to overcome servers' failures, at the same time preserving RYW session guarantee.

[*]This work was supported in part by the State Committee for Scientific Research (KBN), Poland, under grant KBN 3 T11C 073 28

2 BASIC DEFINITIONS

2.1 System Model

Throughout this paper, a replicated, distributed storage system is considered. The system consists of a number of *servers* holding a full copy of a set of data items and *clients* running applications that access these data items. Although all system components (mobile clients, servers, communication links) can be a subject of failures, in this paper we assume that only servers are prone to failures, and thus we do not consider failures of clients and network links. Servers may crash at arbitrary moments and recover after crashing a finite number of times, according to *crash-recovery* model of failures (Guerraoui and Rodrigues, 2004). Clients are mobile, i.e. they can switch from one server to the other. Moreover, they are separated from servers, i.e. a client's application may run on a separate computer than the server. To access shared data, the client selects a single server and sends a direct request to this server. Operations are issued by clients synchronously, i.e. a new operation may be issued after the results of the previous one have been obtained.

The storage replicated by servers does not imply the particular data model or organization. It is a set of data items, which may be simple variables, files, objects of object-oriented programming language, etc. Later in the paper, the data items are referred to as *objects*. Operations performed on shared objects are divided into *reads* and *writes*. Reads do not change the state of objects, while writes does. Some 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 relation $\overset{C_i}{\rightarrow}$ called *client issue order*. As client procecesses are sequential, relation $\overset{C_i}{\rightarrow}$ determines total order on the set of operations issued by C_i. Operations performed by server S_j are ordered by relation $\overset{S_j}{\rightarrowtail}$, called *server execution order*. Since operations are performed one at a time, the server execution order is also totally ordered. Depending on operation type (write, read or these whose type is irrelevant), operations on objects are denoted by w, r or o. The operation performed by server S_j is denoted by $o|_{S_j}$.

2.2 Session Guarantees

Session guarantees have been introduced in the context of Bayou project (Terry et al., 1994). Informally, 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 data storage are monotonic, and finally WFR keeps the track of causal dependencies resulting from operations issued by a client.

In the paper, it is assumed that clients perceive the data from the replicated storage according to Read Your Writes session guarantee. Let us consider some examples, how RYW session guarantee may be used in practice.

First, let us consider a user writing a TODO list to a file. After traveling to another location, the user wants to recall the most urgent tasks, and reads TODO list. Without RYW session guarantee the read may return any previous (possibly empty) version of the document.

Further, imagine that a user, after changing his password, while logging to the system receives an "invalid password" response. This problem would arise in situation when the logging process contacted a server to which a new password had not been propagated. In this case RYW guarantee ensures that the logging process will always read the most recent password. Formally RYW session guarantee is defined as follows (Sobaniec, 2005):

DEFINITION 1. *Read Your Writes (RYW) session guarantee is a property meaning that:*

$$\forall C_i \, \forall S_j \left[w \overset{C_i}{\rightarrow} r|_{S_j} \Rightarrow w \overset{S_j}{\rightarrowtail} r \right]$$

2.3 VsSG Coherency Protocol

Data consistency in the paper is managed by VsSG *consistency protocol* (Kobusińska et al., 2005; Sobaniec, 2005). The VsSG protocol uses a conception of server-based version vectors for efficient representation of sets of writes required by clients. Server-based version vectors have the following form: $V = \left[v_1 \; v_2 \; ... \; v_{N_S} \right]$, 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.

The server, which first obtains the write from the client is responsible for assigning such a write a globally unique identifier, returned by a function $T : \mathcal{O} \mapsto V$ and set to the current value of the vector clock V_{S_j} of server S_j performing the write for the first time. The set of all writes performed by server S_j is denoted by $\mathcal{O}_{S_j}$. The sequence of past writes is called *history* (Sobaniec, 2005). Formally:

DEFINITION 2. *A history H_{S_j} at time moment t, is a linearly ordered set $\left(\mathcal{O}_{S_j}, \overset{S_j}{\rightarrowtail} \right)$ where $\mathcal{O}_{S_j}$ is a set of writes performed by server S_j, till time t and relation $\overset{S_j}{\rightarrowtail}$ represents an execution order of writes.*

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}. Finally, the current value of the server vector clock is returned to the client and causes the update of the client's vector W_{C_i}, representing writes issued by the client. Servers occasionally synchronize states of their replicas by exchanging information about writes performed in the past. As a result, all writes submitted by clients are eventually propagated and executed by every server.

During synchronization of servers, their histories are *concatenated*. The concatenation of histories H_{S_j} and H_{S_k}, denoted by $H_{S_j} \oplus H_{S_k}$ is a sum of writes from the first history and new writes from the second history (Sobaniec, 2005).

2.4 Checkpoint and Log Definitions

Below, we propose formal definitions of mechanisms used by rVsRYW protocol:

DEFINITION 3. *A log Log_{S_j} is a set of triples:*

$$\{ \langle i_1, o_1, T(o_1) \rangle \langle i_2, o_2, T(o_2) \rangle \dots \langle i_n, o_n, T(o_n) \rangle \},$$

where i_n represents the index of the client that issued a write operation o_n, $i_n \in 1..N_C$, and N_C is a number of clients in the system. The operation $o_n \in \mathcal{O}_{S_j}$ and $T(o_n)$ is its timestamp.

During a rollback-recovery procedure, operations from the log are executed according to their timestamps, from the earliest to the latest one.

DEFINITION 4. *A checkpoint $Ckpt_{S_j}$ is a couple $\langle V_{S_j}, H_{S_j} \rangle$, of a version vector V_{S_j} and a history H_{S_j} maintained by server S_j at the time t, where t is a moment of taking a checkpoint.*

For the sake of recovery procedure, it is commonly assumed that servers have access to a stable storage, able to survive all failures (Elmootazbellah et al., 2002). The log and the checkpoint are saved by the server in the stable storage. The newly taken checkpoint replaces the previous one, so just one checkpoint for each server is stored.

3 rVsRYW PROTOCOL

3.1 The General Idea

To preserve RYW session guarantee the protocol must ensure that every write request issued by the client is not lost by the server. Checkpointing every single write operation fulfills this requirement, but results in frequent saving of server state in the stable storage, which is time–consuming. Logging procedure overcomes this disadvantage and takes less time than checkpointing, as only the operation, its timestamp and issuing client index are stored in the stable storage. On the other hand, the log size may grow infinitely and may turn out to be too large. Combining these two approaches by joining logging and checkpointing seems to be the best solution. While applying these known techniques, the semantics of operations, characteristic of session guarantees, is taken into account, which is a novel feature of proposed protocol. Consequently, in rVsSG, only operations essential to provide session guarantees are logged, so checkpoints are optimized with respect to required session guarantee requirements. Moreover, in proposed protocol, it is assumed that only the server, which performs write operation issued directly by a client, logs this request to stable storage. Writes received from other servers during synchronization procedure only cause server's state update, but they are not logged. The checkpoint is taken when the server obtains read operation from a client, and since the latest checkpoint, it has performed any write operation issued directly from this client. Taking a checkpoint results in clearing servers' logs.

After a failure occurrence, the failed server restarts from the latest checkpoint and replays operations from the log to restore the execution to a state that occurred before the failure.

3.2 Protocol Implementation

The request sent from client C_i to server S_j carries the operation that is to be performed and a vector W that is calculated depending on the operation type. W is set to **0** (line 1) or to W_{C_i}(line 3) for writes and reads respectively. Afterwards, the modified message $\langle o, W \rangle$ is send to a server (line 5).

Server S_j obtains operations $\langle o, W, i \rangle$ issued directly by a client or operations $\langle S_k, H \rangle$ issued by other servers in the result of synchronization procedure. Upon receiving a new request from client C_i, S_j checks whether it has performed all writes issued previously by C_i, by comparing version vectors V_{S_j} and W (line 6)(Sobaniec, 2005). If the state of server S_j is not sufficiently up to date, the request is postponed, because RYW session guarantee is not fulfilled (line 7). The request will be resumed after synchronization with another server (line 40).

When the client requests to perform a write operation (line 9), then server S_j stores issuing client identifier by adding clients' index to the set CW_{S_j} (line 10). Further S_j updates its data structures: increases the value of its version vector V_{S_j} and timestamps the operation o, to give o a unique identifier (lines 11-12). Before performing operation o, S_j logs data necessary to recover its state in case the failure occurrence (line 13). When the information necessary for rollback-recovery is logged, the server performs the client's request (line 14) and adds it to its history of performed writes (line 15).

When the read request from client C_i is received by server S_j, the server checks first if, since the latest checkpoint, it has performed any write operation submitted by C_i (line 17). If not, the previously taken checkpoint possesses all information necessary to guarantee RYW to C_i and the new checkpoint does not need to be taken. Otherwise, when at least one write request issued by C_i has been performed by S_j, the state of the server is checkpointed (line 19).

Checkpointing the server state causes clearing log Log_{S_j} and set CW_{S_j} (lines 20-21).

After the failure, the server state is recovered according to the information remembered in the latest checkpoint $Ckpt_{S_j}$ of server S_j (line 41) and in log Log_{S_j} (lines 42-50). Recovered operations are added to the vectors V_{S_j} (line 46) and CW_{S_j} (line 49), as well as to history H_{S_j} (line 48).

During rollback-recovery procedure it is important that logging of write operations takes place before performing them. Such an order is crucial because, if the operation is performed but not logged, it could be lost in the case of failure. When failures happen during the rollback-recovery procedure, the recovery action is just prolonged, as the server must be rolled back again and operations from the log have to be executed from the beginning. The repeated recovery procedure works correctly, as the content of the checkpoint and the log is not changed.

4 PROOF OF SAFETY PROPERTY

The safety property asserts that clients access object replicas maintained by servers according to RYW session guarantee, regardless of servers' failures.

For the sake of the proof simplicity we introduce some auxiliary lemmas. First, we state that every write operation issued by a client and performed by a server is not lost in the rVsRYW 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 a 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} (line 13). Because logging of w takes place before performing it (line 14), then even in the case of failure the operation w is not lost, but remains in the log.
2. Log Log_{S_j} is cleared after performing by S_j the read operation. However, according to the algorithm, reads cause storing the information on writes by checkpointing the server's version vector V_{S_j} and history H_{S_j} in $Ckpt_{S_j}$ (line 19). The checkpoint is taken before the operation of clearing log Log_{S_j} (line 20). Therefore, the server failure that occurs after clearing the log does not affect safety of the algorithm, because writes from the log are already stored in the checkpoint.

```
Upon sending a request $\langle o \rangle$ to server $S_j$ at client $C_i$
 1: $W \leftarrow \mathbf{0}$
 2: if (not iswrite($o$)) then
 3:     $W \leftarrow W_{C_i}$
 4: end if
 5: send $\langle o, W \rangle$ to $S_j$

Upon receiving a request $\langle o, W \rangle$ from client $C_i$ at server $S_j$
 6: while $\left(V_{S_j} \ngeq W\right)$ do
 7:     wait()
 8: end while
 9: if iswrite($o$) then
10:     $CW_{S_j} \leftarrow CW_{S_j} \cup i$
11:     $V_{S_j}[j] \leftarrow V_{S_j}[j] + 1$
12:     timestamp $o$ with $V_{S_j}$
13:     $Log_{S_j} \leftarrow Log_{S_j} \cup \langle i, o, T(o) \rangle$
14:     perform $o$ and store results in $res$
15:     $H_{S_j} \leftarrow H_{S_j} \oplus \{o\}$
16: end if
17: if (not iswrite($o$)) then
18:     if $i \in CW_{S_j}$ then
19:         $Ckpt_{S_j} \leftarrow \langle V_{S_j}, H_{H_j} \rangle$
20:         $Log_{S_j} \leftarrow \emptyset$
21:         $CW_{S_j} \leftarrow \emptyset$
22:     end if
23:     perform $o$ and store results in $res$
24: end if
25: 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 $C_i$
26: if iswrite($o$) then
27:     $W_{C_i} \leftarrow \max\left(W_{C_i}, W\right)$
28: end if
29: deliver $\langle res \rangle$

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

Upon receiving an update $\langle S_k, H \rangle$ at server $S_j$
33: foreach $w_i \in H$ do
34:     if $V_{S_j} \ngeq T(w_i)$ then
35:         perform $w_i$
36:         $V_{S_j} \leftarrow \max\left(V_{S_j}, T(w_i)\right)$
37:         $H_{S_j} \leftarrow H_{S_j} \oplus \{w_i\}$
38:     end if
39: end for
40: signal()

On rollback-recovery
41: $\langle V_{S_j}, H_{S_j} \rangle \leftarrow Ckpt_{S_j}$
42: $vrecover \leftarrow \mathbf{0}$
43: foreach $o'_j \in Log'_{S_j}$ do
44:     choose $\langle i', o'_i, T(o'_i) \rangle$ with minimal $T(o'_j)$
45:      from $Log'_{S_j}$ where $T(o'_j) > V_{S_j}$
46:     $V_{S_j}[j] \leftarrow V_{S_j}[j] + 1$
47:     perform $o'_j$
48:     $H_{S_j} \leftarrow H_{S_j} \oplus \left\{o'_j\right\}$
49:     $CW_{S_j} \leftarrow CW_{S_j} \cup i'$
50:     $vrecover \leftarrow T(o'_i)$
51: end for
```

3. After the checkpoint is taken, but before the log is cleared (between lines 19 and 20) writes issued by client C_i and performed by server S_j are stored in both the checkpoint $Ckpt_{S_j}$ and the log Log_{S_j}.

Hence, every write operation w before being performed, but not yet checkpointed, is stored in the log Log_{S_j} (from 1). At the moment of clearing the log, all operations from the log are already saved in the checkpoint $Ckpt_{S_j}$ (from 2). There is a time, where operations performed by S_j are saved in both data structures: the checkpoint and the log (from 3). Data stored in the log and the checkpoint are kept in the stable storage. As a result, even in the case of server S_j failure the write w received by S_j directly from client C_i is not lost and is stored either in the log or in the checkpoint, or in both these structures. □

The checkpoint definition (Definition 4) leads us to the observation, that all write operations issued by client C_i and performed by server S_j, that have been checkpointed in the latest checkpoint $Ckpt_{S_j}$ taken before the failure of S_j, are recovered during the rollback-recovery procedure.

In contrast to recovery of checkpointed operations (line 41), the recovery of operations saved in the log is not made atomically (lines 43-49). Therefore, there is a possibility that server failure may occur in the middle of recovery procedure, when some logged operations have been recovered, but others have not. Thus, the recovery of all logged operations is not obvious. For that reason below, in Lemma 2, we prove that also all operations logged in log Log_{S_j}, which have been performed after the moment of taking checkpoint but before the failure of S_j, are recovered during recovery procedure.

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

Proof. According to the algorithm, all write operations issued by client C_i and performed by server S_j are logged in log Log_{S_j} before being performed (line 13).

Let us assume server S_j fails. The rollback-recovery procedure after recovering values V_{S_j} and H_{S_j} from a checkpoint, recovers all operations (line 43) performed by S_j before the failure occurred, but after the checkpoint was taken. These operations are recovered due to values remembered in the log (line 44) — from the log definition these are the issuing client index, the operation and its timestamp. The recovered operation updates version vector V_{S_j} (line 46), it is performed by S_j (line 47) and added to the server's S_j history H_{S_j}(line 48).

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 20) and it is not modified during the rollback-recovery procedure, the log's content is not changed. Hence, the recovery procedure can be started from the beginning without loss of any operation issued by client C_i and performed by server S_j after the moment of taking checkpoint. □

To preserve RYW session guarantee, all write operations performed by server S_j have to be recovered before new read operation is obtained from client C_i. Otherwise, there is a possibility that some writes previously issued by C_i are still not recovered and the read operation cannot be performed according to RYW.

LEMMA 3. *The server performs new read operation issued by a client only after all writes performed before the failure are recovered.*

Proof. By contradiction, let us assume that there is write operation w performed by server S_j before the failure occurrence, that has not been recovered yet, and that the server has performed a new read operation issued by client C_i. According to underlying VsSG protocol, for server S_j that performs the new read operation, the condition

$$V_{S_j} \geq W_{C_i}$$

is fulfilled (lines 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 (line 12). The server that has performed write sends a reply that contains the modified vector V_{S_j} to the client.

At the client side, after the reply is received, vector W_{C_i} is modified: $W_{C_i} \leftarrow \max(W, W_{C_i})$. This means that vector W_{C_i} is updated at least at position j: $W_{C_i}[j] \leftarrow \max[j] + 1$.

If there is a write operation w performed by server S_j before the failure that has not been recovered yet, then $V_{S_j}[j] < W_{C_i}[j]$, which follows from the ordering of recovered operations (line 44). This is a contradiction with $V_{S_j} \geq W_{C_i}$. Hence, the read operation cannot be performed until all previous writes are recovered. □

THEOREM 1. *RYW session guarantee is preserved by rVsRYW protocol for clients requesting it, even in the presence of server failures.*

Proof. Let us consider operations w and r, issued by client C_i, which requires RYW session guarantee. Let read operation follows a write one in the client's issue order and let read be performed by server S_j.

It has been proven that VsSG protocol preserves RYW session guarantee, when none of servers fails, i.e. for any client C_i requiring RYW and for any server S_j the relation $\forall C_i \, \forall S_j \left[w \overset{C_i}{\rightarrow} r|_{S_j} \Rightarrow w \overset{S_j}{\rightarrowtail} r \right]$ holds. According to Lemma 1, every write operation performed by server S_j is saved in the checkpoint or in the log. After the server 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 3, all recovered operations are applied before new ones.

Hence, for any client C_i and any server S_j, RYW session guarantee is preserved by the rollback- recovery and checkpointing rVsRYW algorithm. $\square$

5 CONCLUSIONS

The growing attention paid to mobile systems, results in their increased reliability requirements. Most of rollback-recovery protocols which take into consideration characteristics of mobile environment, pay special attention to increasing efficiency of the protocol. The efficiency is usually increased through minimization of the amount of messages exchanged between mobile hosts during taking a checkpoint. On the other hand, the protocol efficiency can also be increased by data replication. However, in this approach the problem of replica consistency during the recovery process is faced and should be solved. According to our knowledge, although several studies have examined the issues of checkpointing, logging and rollback-recovery in mobile environment, none of the existing solutions integrates these issues with the consistency protocols. Especially client-centric consistency models, regarding consistency from the client's point of view, have not been considered in the context of rollback-recovery.

Therefore, this paper addresses a problem of integrating the consistency management of mobile systems with the recovery mechanisms. We introduce rVsRYW rollback-recovery protocol for distributed mobile systems, which provides Read Your Writes session guarantee. The proposed recovery protocol is integrated with the underlying VsSG consistency protocol. Additionally, the proof of safety property of rVsRYW protocol is included.

The rVsRYW protocol, takes into account the semantics of operations during the rollback-recovery procedure. This results in checkpointing only results of write operations. As a result, rVsRYW protocol offers the ability to overcome the servers' failures and ensures RYW session guarantee, in the optimized way.

REFERENCES

Alvasi, L. and Marzullo, K. (1998). Message logging: pessimistic, optimistic, causal and optimal. *IEEE Trans. Softw. Eng*, 24(2):149–159.

Bernstein, P. A., Hadzilacos, V., and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*. Addison Wesley.

Duchamp, D., Feiner, S., and Jr, G. M. (1991). Software technology for wireless mobile computing. *IEEE Network Magazine*, pages 2–18.

Elmootazbellah, N., Elnozahy, Lorenzo, A., Wang, Y.-M., and Johnson, D. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408.

Elnozahy, E. and Zwaenepoel, W. (1992). Manetho: Transparent rollback-recovery with low overhead, limited rollback, and fast output commit. *IEEE Transactions on Computer*, 41(5):526–531.

Guerraoui, R. and Rodrigues, L. (2004). *Introduction to distributed algorithms*. Springer-Verlag.

Kobusińska, A., Libuda, M., Sobaniec, C., and Wawrzyniak, D. (2005). Version vector protocols implementing session guarantees. *Proc. of Int. Symp. on Cluster Computing and the Grid (CCGrid 2005)*.

Pradhan, D., P.Krishna, and Vaidya, N. (1996). Recovery in mobile environments: Design and trade-off analysis. *Proc. of the 26th International Symposium on Fault-Tolerant Computing*, pages 16–25.

Sergent, N., Dfago, X., and Schiper, A. (1999). Failure detectors: Implementation issues and impact on consensus performance. Technical Report SSC/1999/019, cole Polytechnique Fdrale de Lausanne, Switzerland.

Sobaniec, C. (2005). *Consistency Protocols of Session Guarantees in Distributed Mobile Systems*. PhD thesis, Institute of Computing Science, Poznan University of Technology.

Szychowiak, M. (2003). *Replication of checkpoints in DSM systems with read-write objects*. PhD thesis, Institute of Computing Science, Poznan University of Technology.

Tanaka, K., Higaki, H., and Takizawa, M. (1998). Object-based checkpoints in distributed systems. *Journal of computer system science and Engineering*, 13(3):125–131.

Terry, D. B., Demers, A. J., Petersen, K., Spreitzer, M., Theimer, M., and Welch, B. W. (1994). Session guarantees for weakly consistent replicated data. *Proc. of the Third Int. Conf. on Parallel and Distributed Information Systems (PDIS 94)*, pages 140–149.