Safety of a Server-Based Version Vector Protocol Implementing Session Guarantees^{*}

Jerzy Brzeziński, Cezary Sobaniec, and Dariusz Wawrzyniak

Institute of Computing Science,
Poznań University of Technology, Poland

{Jerzy.Brzezinski, Cezary.Sobaniec, Dariusz.Wawrzyniak}@cs.put.poznan.pl

Abstract. Session guarantees are used to manage replica consistency of a distributed system from the client perspective. This paper defines formally the guarantees, presents and proves safety of a protocol implementing session guarantees using server-based version vectors.

1 Introduction

Replication is a key concept in providing high performance and availability of data and services in a distributed system. However, replication introduces the problem of data consistency that arises when replicas are modified. Required properties of distributed system with respect to consistency depend in general on application and are formally specified by *consistency models*. There are numerous consistency models developed for *Distributed Shared Memory* systems. These models, called *data-centric* consistency models [1], assume that servers replicating data are also accessing the data for processing purposes. In a mobile environment, however, clients accessing the data are not bound to particular servers, they can switch from one server to another. This switching adds a new dimension of complexity to the problem of consistency. *Session guarantees* [2], called also *client-centric* consistency models [1], have been proposed to define required properties of the system regarding consistency from the client's point of view. Intuitively: the client wants to continue processing after a switch to another server so that new operations will remain consistent with previously issued operations within a *session*. The relationships between data-centric and client-centric consistency models have been analyzed in [3, 4]. Protocols implementing session guarantees must efficiently represent sets of operations performed in the system. Version vectors based on vector clocks [5] may be used for this purpose. This paper presents a protocol, called VsSG, implementing session guarantees, that uses server-based version vectors. The protocol is based on a concept presented in [2]. The main contribution of this paper is a formal proof of safety of the VsSG protocol. The formalism introduced in this paper includes also formal definitions of session guarantees.

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

2 Session Guarantees

In this paper we consider a weakly consistent replicated storage system. The system consists of a number of *servers* holding a full copy of a set of *shared objects*, and *clients* running applications that access the objects. Clients are separated from servers, i.e. a client application may run on a separate computer than the server. A client may access a shared object after selecting a single server and sending a direct request to the server. Clients are mobile, i.e. they can switch from one server to another during application execution. Session guarantees are expected to take care of data consistency observed by a migrating client. The set of shared objects replicated by the servers does not imply any particular data model or organization. Operations performed on shared objects are divided into *reads* and *writes*. A read does not change states of the shared objects, while a write does. A write may cause an update of an object, it may create a new object, or delete an existing one. A write may also atomically update states of several objects.

Operations on shared objects issued by a client C_i are ordered by a relation $\xrightarrow{C_i}$ called *client issue order*. A server S_j performs operations in an order represented by a relation $\xrightarrow{S_j}$. Writes and reads on objects will be denoted by w and r , respectively. An operation performed by a server S_j will be denoted by $w|_{S_j}$ or $r|_{S_j}$.

Definition 1. *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 .*

The exact meaning of *relevant writes* will strongly depend on the characteristics of a given system or application. For example, in case of simple isolated objects (i.e. objects with methods that access only their internal fields), relevant writes of a read on object x may be represented by all previous writes on object x .

Session guarantees have been defined in [2]. The following more formal definitions are based on those concepts. The definitions assume that operations are unique, i.e. they are labeled by some internal unique identifiers.

Definition 2. *Read Your Writes (RYW) session guarantee is defined as follows:*

$$\forall C_i \forall S_j \left[w \xrightarrow{C_i} r|_{S_j} \Rightarrow w \xrightarrow{S_j} r \right]$$

Definition 3. *Monotonic Writes (MW) session guarantee is defined as follows:*

$$\forall C_i \forall S_j \left[w_1 \xrightarrow{C_i} w_2|_{S_j} \Rightarrow w_1 \xrightarrow{S_j} w_2 \right]$$

Definition 4. *Monotonic Reads (MR) session guarantee is defined as follows:*

$$\forall C_i \forall S_j \left[r_1 \xrightarrow{C_i} r_2|_{S_j} \Rightarrow \forall w_k \in RW(r_1) : w_k \xrightarrow{S_j} r_2 \right]$$

Definition 5. *Writes Follow Reads (WFR) session guarantee is defined as follows:*

$$\forall C_i \forall S_j \left[r \xrightarrow{C_i} w|_{S_j} \Rightarrow \forall w_k \in RW(r) : w_k \xrightarrow{S_j} w \right]$$

3 The VsSG Protocol Implementing Session Guarantees

The proposed VsSG protocol implementing session guarantees intercepts communication between clients and servers; at the client side before sending a request, at the server side after receiving the request and before sending a reply, and at the client side after receiving the reply. These interceptions are used to exchange and maintain additional data structures necessary to preserve appropriate session guarantees. After receipt of a new request a server checks whether its state is sufficiently up to date to satisfy client's requirements. If the server's state is outdated then the request is postponed and will be resumed after updating the server.

Servers occasionally exchange information about writes performed in the past in order to synchronize the states of replicas. This synchronization procedure eventually causes total propagation of all writes directly submitted by clients. It does not influence safety of the VsSG protocol but rather its liveness, therefore it will not be discussed in this paper (example procedure is presented in [6]). In contrast with [2] we do not assume total ordering of non-commutative writes which is treated by us as an orthogonal problem.

Every server S_j records all writes performed locally in a history. The writes result from direct client requests, or are incorporated from other servers during synchronization procedure. The writes are performed sequentially, therefore the history is totally ordered. Formally, histories are defined as follows:

Definition 6. *A history H_{S_j} is a linearly ordered set $\left(\mathcal{O}_{S_j}, \xrightarrow{S_j} \right)$ where $\mathcal{O}_{S_j}$ is a set of writes performed by a server S_j , and relation $\xrightarrow{S_j}$ represents an execution order of the writes.*

During synchronization of servers the histories are *concatenated*. Intuitively, a concatenation of two histories is constructed as a sum of the first history, and new writes found in the second history. The orderings of respective histories are preserved, and new writes are added at the end of the first history.

Server-based version vectors. Version vectors are used for efficient representation of sets of writes required by clients and necessary to check at the server side. Version vectors used in this paper have the following form: $[v_1 \ v_2 \ \dots \ v_{N_S}]$, where N_S is the total number of servers in the system. A single position v_j denotes the number of writes performed by a server S_j , and changes whenever a new write request is performed by the server. Because every server increments the version vector for every write, and the changes are done at different positions,

the values of version vectors at servers during execution of writes are unique. A write in VsSG protocol is labeled with a vector timestamp set to the current value of the version vector V_{S_j} of the server S_j performing the write for the first time. In the presentation of the VsSG protocol the vector timestamp of a write w is returned by a function $T : \mathcal{O} \mapsto V$. A single i -th position of the version vector timestamp associated with a write will be denoted by $T(w)[i]$.

The VsSG protocol (presented in Alg. 1) interacts with requests sent from clients to servers and with replies sent from servers to clients. A request is a couple $\langle op, SG \rangle$, where op is an operation to be performed, and SG is a set of session guarantees required for this operation. Before sending to the server, the request is supplemented with a vector W representing the client's requirements. A reply is a triple $\langle op, res, W \rangle$ where op is the operation just performed, res represents the results of the operation (delivered to the application), and W is a vector representing the state of the server just after performing the operation.

Before sending a request by a client C_i , a vector W representing its requirements is calculated based on the type of operation, and the set SG of session guarantees required for the operation. The vector W is set to either $\mathbf{0}$, or W_{C_i} — a vector representing writes issued by the client C_i , or R_{C_i} — a vector representing writes relevant to reads issued by the client, or to a maximum of these two vector (lines 1, 3 and 6). The maximum of two vectors V_1 and V_2 is a vector $V = \max(V_1, V_2)$, such that $V[i] = \max(V_1[i], V_2[i])$.

On receipt of a new request a server S_j checks whether its local version vector V_{S_j} *dominates* the vector W sent by the client (line 9), which is expected to be sufficient for providing appropriate session guarantees. A version vector V_1 dominates a version vector V_2 , which is denoted by $V_1 \geq V_2$, when $\forall i : V_1[i] \geq V_2[i]$. If the state of the server is not sufficiently up to date, the request is postponed (line 10), and will be resumed after synchronization with another server (line 32). As a result of writes performed by a server S_j , its version vector V_{S_j} is incremented at position j (line 14), and a timestamped operation is recorded in history H_{S_j} (lines 15 and 16). The current value of the server version vector V_{S_j} is returned to the client (line 18) and updates the client's vector W_{C_i} in case of writes (line 20), or R_{C_i} in case of reads (line 22).

4 Safety of the VsSG Protocol

Definition 7. A supremum of a set of writes O_{S_j} , denoted by $\overline{V}(O_{S_j})$, is a vector that is set to $\mathbf{0}$ for an empty set, and for nonempty sets its i -th position is defined as $\overline{V}(O_{S_j})[i] = \max_{w \in O_{S_j}} T(w)[i]$.

Lemma 1. For every server S_j running VsSG protocol at every moment $\overline{V}(O_{S_j}) = V_{S_j}$.

Proof. By induction. 1) Basis. At the very beginning $V_{S_j} = \mathbf{0}$, and the set of writes $O_{S_j} = \emptyset$, therefore $\overline{V}(O_{S_j}) = \mathbf{0}$, hence $\overline{V}(O_{S_j}) = V_{S_j}$. 2) Induction step.

```

On send of request message  $\langle op, SG \rangle$  from  $C_i$  to  $S_j$ 
1:  $W \leftarrow 0$ 
2: if (iswrite( $op$ ) and  $MW \in SG$ ) or (not iswrite( $op$ ) and  $RYW \in SG$ ) then
3:    $W \leftarrow \max(W, W_{C_i})$ 
4: end if
5: if (iswrite( $op$ ) and  $WFR \in SG$ ) or (not iswrite( $op$ ) and  $MR \in SG$ ) then
6:    $W \leftarrow \max(W, R_{C_i})$ 
7: end if
8: send  $\langle op, W \rangle$  to  $S_j$ 

On receipt of request message  $\langle op, W \rangle$  from client  $C_i$  at server  $S_j$ 
9: while ( $V_{S_j} \not\geq W$ ) do
10:   wait
11: end while
12: perform  $op$  and store results in  $res$ 
13: if iswrite( $op$ ) then
14:    $V_{S_j}[j] \leftarrow V_{S_j}[j] + 1$ 
15:   timestamp  $op$  with  $V_{S_j}$ 
16:    $H_{S_j} \leftarrow H_{S_j} \cup \{op\}$ 
17: end if
18: send  $\langle op, res, V_{S_j} \rangle$  to  $C_i$ 

On receipt of reply message  $\langle op, res, W \rangle$  from server  $S_j$  at client  $C_i$ 
19: if iswrite( $op$ ) then
20:    $W_{C_i} \leftarrow \max(W_{C_i}, W)$ 
21: else
22:    $R_{C_i} \leftarrow \max(R_{C_i}, W)$ 
23: end if
24: deliver  $\langle res \rangle$ 

On receipt of update message  $\langle S_k, H \rangle$  at server  $S_j$ 
25: foreach  $w_i \in H$  do
26:   if  $V_{S_j} \not\geq T(w_i)$  then
27:     perform  $w_i$ 
28:      $V_{S_j} \leftarrow \max(V_{S_j}, T(w_i))$ 
29:      $H_{S_j} \leftarrow H_{S_j} \cup \{w_i\}$ 
30:   end if
31: end for
32: signal

Every  $\Delta t$  at server  $S_j$ 
33: foreach  $S_k \neq S_j$  do
34:   send  $\langle S_j, H_{S_j} \rangle$  to  $S_k$ 
35: end for

```

Algorithm 1. VsSG protocol implementing session guarantees

Let us assume a state where condition $\overline{V}(O_{S_j}) = V_{S_j}$ holds. The set O_{S_j} and the version vector V_{S_j} can change only in the following two situations:

- The server S_j accepts a new write requested by a client. This causes the value of $V_{S_j}[j]$ to be incremented by 1, next the write is timestamped with the current value of vector V_{S_j} , and the write is added to O_{S_j} . This causes $\overline{V}(O_{S_j})$ to be also incremented at position j by 1 (lines 14 and 16 of Alg. 1). As a result the condition $\overline{V}(O_{S_j}) = V_{S_j}$ still holds.
- The server S_j incorporates a write w received from another server. This causes the current value of V_{S_j} to be maximized with the vector $T(w)$ of the write being added (line 28). The new write is then added to O_{S_j} (line 29). As a result values of V_{S_j} and $\overline{V}(O_{S_j})$ will be incremented at the same positions by the same values, therefore the condition $\overline{V}(O_{S_j}) = V_{S_j}$ still holds. $\square$

Definition 8. A write-set $WS(V)$ of a given version vector V is defined as $WS(V) = \bigcup_{j=1}^{N_S} \{w \in O_{S_j} : T(w) \leq V\}$.

Lemma 2. For any two vectors V_1 and V_2 used by servers and clients of the VsSG protocol $V_1 \geq V_2 \Leftrightarrow WS(V_1) \supseteq WS(V_2)$.

Proof. 1) Sufficient condition. By contradiction, let us assume that $V_1 \geq V_2 \wedge WS(V_1) \not\supseteq WS(V_2)$, which means that $\exists w [w \notin WS(V_1) \wedge w \in WS(V_2)]$ and, according to Definition 8: $\exists j (T(w)[j] > V_1[j] \wedge T(w)[j] \leq V_2[j]) \Rightarrow V_1[j] < V_2[j] \Rightarrow V_1 \not\geq V_2$. 2) Necessary condition. By contradiction, let us assume that $WS(V_1) \supseteq WS(V_2) \wedge V_1 \not\geq V_2$, which means that $\exists j : V_1[j] < V_2[j]$. Version vectors at position j are only incremented when a new write is performed by a server S_j (line 14). It means that $\exists w \in O_{S_j} [w \in WS(V_2) \wedge w \notin WS(V_1)]$ and hence $WS(V_1) \not\supseteq WS(V_2)$. $\square$

Lemma 3. At any time during execution of VsSG protocol $O_{S_j} = WS(V_{S_j})$.

Proof. By contradiction: 1) Let us assume that $\exists w \in O_{S_j} : w \notin WS(V_{S_j})$. According to Definition 8, a write w does not belong to $WS(V_{S_j})$ when $T(w) \not\leq V_{S_j}$. This implies that $\exists k : T(w)[k] > V_{S_j}[k]$, and, according to Lemma 1, $T(w)[k] > \overline{V}(O_{S_j})[k]$, which implies $\overline{V}(O_{S_j}) \not\leq T(w)$. Based on Definition 7, $w \notin O_{S_j}$ — a contradiction. 2) Let us assume that $\exists w \in WS(V_{S_j}) : w \notin O_{S_j}$. According to Definition 7, a write w does not belong to O_{S_j} when $\overline{V}(O_{S_j}) \not\leq T(w)$. This implies that $\exists k : T(w)[k] > \overline{V}(O_{S_j})[k]$, and, according to Lemma 1, $T(w)[k] > V_{S_j}[k]$, which implies $T(w) \not\leq V_{S_j}$. Based on Definition 8, $w \notin WS(V_{S_j})$ — a contradiction. $\square$

Lemma 4. At any time during execution of VsSG protocol $WS(W_{C_i})$ contains all writes issued by a client C_i .

Proof. A write issued by a client C_i and performed by a server S_j updates the client's vector W_{C_i} by calculating a maximum of its current value and value of

the server version vector V_{S_j} (lines 18 and 20). Hence, after performing the write $W_{C_i} \geq V_{S_j}$, and (according to Lemma 2) $WS(W_{C_i}) \supseteq WS(V_{S_j})$, and (according to Lemma 3) $WS(W_{C_i}) \supseteq O_{S_j}$. It means that the write-set $WS(W_{C_i})$ contains all writes requested directly at server S_j , including also writes requested by the client C_i at server S_j . The vector W_{C_i} monotonically increases, therefore no past write is lost in case of a migration to another server. $\square$

Lemma 5. *At any time during execution of VsSG protocol $WS(R_{C_i})$ contains all writes relevant to reads issued by a client C_i .*

Proof. A read issued by a client C_i and performed by a server S_j updates the client's vector R_{C_i} by calculating a maximum of its current value and value of the server version vector V_{S_j} (lines 18 and 22). Hence (according to Lemmata 2 and 3) $R_{C_i} \geq V_{S_j} \Rightarrow WS(R_{C_i}) \supseteq WS(V_{S_j}) = O_{S_j}$. It means that the write-set $WS(R_{C_i})$ contains all writes performed at server S_j , therefore also writes relevant to reads requested by the client C_i at server S_j . The vector R_{C_i} monotonically increases, therefore no past write is lost in case of a migration to another server. $\square$

Theorem 1. *RYW session guarantee is preserved by VsSG protocol for clients requesting it.*

Proof. Let us consider two operations w and r , issued by a client C_i requiring RYW session guarantee. Let the read follow the write in the client's issue order, and let the read be performed by a server S_j , i.e. $w \xrightarrow{C_i} r|_{S_j}$. After performing w we have (according to Lemma 4) $w \in WS(W_{C_i})$. Because $V_{S_j} \geq W_{C_i}$ is fulfilled before performing r (lines 3 and 9), we get (according to Lemma 2) $WS(V_{S_j}) \supseteq WS(W_{C_i}) \Rightarrow w \in WS(V_{S_j})$. Because local operations at servers are totally ordered, we get $w \xrightarrow{S_j} r$. This will happen for any client C_i requiring RYW and any server S_j , so $\forall C_i \forall S_j \left[w \xrightarrow{C_i} r|_{S_j} \Rightarrow w \xrightarrow{S_j} r \right]$, which means that RYW session guarantee is preserved. $\square$

Theorem 2. *MR session guarantee is preserved by VsSG protocol for clients requesting it.*

Proof. Let us consider two reads r_1 and r_2 , issued by a client C_i requiring MR session guarantee. Let the second read follow the first read in the client's issue order, and let the second read be performed by a server S_j , i.e. $r_1 \xrightarrow{C_i} r_2|_{S_j}$. After performing r_1 we have (according to Lemma 5) $\forall w_k \in RW(r_1) : w_k \in WS(R_{C_i})$. Because $V_{S_j} \geq R_{C_i}$ is fulfilled before performing r_2 (lines 6 and 9), we get (according to Lemma 2) $WS(V_{S_j}) \supseteq WS(R_{C_i}) \Rightarrow \forall w_k \in RW(r_1) : w_k \in WS(V_{S_j})$. Because local operations at servers are totally ordered, we get $\forall w_k \in RW(r_1) : w_k \xrightarrow{S_j} r_2$. This will happen for any client C_i and any server S_j , so $\forall C_i \forall S_j \left[r_1 \xrightarrow{C_i} r_2|_{S_j} \Rightarrow \forall w_k \in RW(r_1) : w_k \xrightarrow{S_j} r_2 \right]$, which means that MR session guarantee is preserved. $\square$

A theorem and a proof for MW are analogous to RYW. A theorem and a proof for WFR are analogous to MR. Full versions of the theorems and proofs can be found in [7].

5 Conclusions

This paper has presented formal definitions of session guarantees, a VsSG protocol implementing session guarantees, and finally a correctness proof showing that the protocol is safe, i.e. appropriate guarantees are provided. It is worth mentioning, however, that though the server-based version vectors used in the VsSG protocol are sufficient for fulfilling session guarantees, they are not necessary. Thus, other approaches are also possible, and they have been discussed in [8]. The sets of writes represented by version vectors are supersets of the exact sets resulting from appropriate definitions. The accuracy of the write-set representation is therefore an important factor of a protocol implementing session guarantees influencing its performance. This problem is currently considered, and appropriate simulation experiments are being prepared.

References

1. Tanenbaum, A.S., van Steen, M.: *Distributed Systems — Principles and Paradigms*. Prentice Hall, New Jersey (2002)
2. Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M., Theimer, M., Welch, B.W.: Session guarantees for weakly consistent replicated data. In: *Proceedings of the Third International Conference on Parallel and Distributed Information Systems (PDIS 94)*, Austin, Texas, September 28-30, 1994, IEEE Computer Society (1994) 140–149
3. Brzeziński, J., Sobaniec, C., Wawrzyniak, D.: Session guarantees to achieve PRAM consistency of replicated shared objects. In: *Proc. of Fifth Int. Conference on Parallel Processing and Applied Mathematics (PPAM'2003)*, LNCS 3019, CzOstochowa, Poland (2003) 1–8
4. Brzeziński, J., Sobaniec, C., Wawrzyniak, D.: From session causality to causal consistency. In: *Proc. of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP2004)*, A Coruña, Spain (2004) 152–158
5. Mattern, F.: Virtual time and global states of distributed systems. In: Cosnard, Quinton, Raynal, Robert, eds.: *Proc. of the Int'l. Conf. on Parallel and Distributed Algorithms*, Elsevier Science Publishers B. V. (1988) 215–226
6. Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M., Demers, A.J.: Flexible update propagation for weakly consistent replication. In: *Proc. of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, Saint Malo, France (1997) 288–301
7. Brzeziński, J., Sobaniec, C., Wawrzyniak, D.: Safety of VsSG protocol implementing session guarantees. Technical Report RA-003/05, Institute of Computing Science, Poznań University of Technology (2005)
8. Kobusińska, A., Libuda, M., Sobaniec, C., Wawrzyniak, D.: Version vector protocols implementing session guarantees. In: *Proc. of Int'l Symposium on Cluster Computing and the Grid (CCGrid 2005)*, Cardiff, UK (2005)