Control and Cybernetics

vol. **40** (2011) No. 3

Performance evaluation of different types of session guarantees version vectors*†

by

Łukasz Piątkowski, Cezary Sobaniec and Grzegorz Sobański

Poznań University of Technology, Institute of Computing Science
Poznań, Poland

{Lukasz.Piatkowski, Cezary.Sobaniec, Grzegorz.Sobanski}@cs.put.poznan.pl

Abstract: Session guarantees define consistency properties of replicas in a distributed system from the point of view of a single, migrating client. Consistency is checked by ensuring execution of required operations by servers. Consistency protocols of session guarantees usually use version vectors for efficient representation of sets of writes. This paper presents performance evaluation of several consistency protocols of session guarantees that use different forms of version vectors. The evaluation was carried out by means of simulation experiments.

Keywords: session guarantees, distributed, replication.

1. Introduction

For many years distributed systems have been gaining increasing popularity and have become a standard for delivering services. With a decreasing cost of mobile devices like laptops, PDAs, smartphones and common availability of wireless networks, a distributed environment becomes a commonplace for end-users. They want to access their data anytime from anywhere. Moreover, they want this access to be highly effective and interactive.

Wireless mobile systems are also gaining importance in fields of sensor and mobile ad-hoc networks. In these environments the problem of data replication and consistency is particularly hard. Still, there are some cases where mobile ad-hoc networks are the only solution available, e.g. in dynamic search, rescue or evacuation missions, where there is no time to deploy network infrastructure and existing wireless networks may be damaged or unavailable.

One of the methods of satisfying needs of mobile applications is to use replication to provide fast and reliable access to data. The drawback of that approach

*Submitted: March 2011; Accepted: August 2011.

†The research presented in this paper has been partially supported by the European Union within the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

is the problem of data consistency, which arises when replicas are modified. In a mobile environment clients are free to change their location, thus changing the network node and topology at which they access data.

Numerous consistency models were developed for *Distributed Shared Memory* systems (DSM). These models, called *data-centric* consistency models (Tanenbaum & van Steen, 2002), assume that processes replicating data are also accessing the data. All consistency requirements are defined from the point of view of a data object.

Another approach, designed with mobile environments in mind, are *session guarantees* (Terry et al., 1994), also called *client-centric* consistency models (Tanenbaum & van Steen, 2002). They have been proposed to define required properties of consistency from the point of view of a single migrating client. In this approach clients explicitly indicate which operations are important for them, i.e. which operations should be synchronized after switching to another server. The sets of writes are monotonically growing as the client is requesting new operations. Meanwhile, other clients may perform some new operations at the same server, so the client also learns about these updates.

Protocols implementing session guarantees must efficiently represent sets of operations performed in the system. Version vectors based on vector clocks (Mattern, 1998; Fidge, 1991) may be used for this purpose. The original protocol presented in Terry et al. (1994) used server-based version vectors, where each position of the version vector represents the number of writes performed by a given server. It has been shown that other approaches are also possible: protocols using client-based or object-based version vectors (Kobusińska et al., 2005). Positions in such vectors represent respectively: the number of writes performed by the clients or the number of writes performed on given objects. There are also variants of version vectors like dynamic sized vectors (Ratner et al., 1997) or plausible clocks (Torres-Rojas & Ahamad, 1999; Brzeziński et al., 2007). The choice of version vector type depends strongly on specificity of the application.

One of the key elements of a consistency protocol providing session guarantees is a server synchronization protocol. For early performance evaluation a very simple, periodically broadcasting server synchronization method was used (Sobaniec, 2005). To overcome its limitations, new synchronization protocols were proposed: *On-Demand Synchronization Algorithm with Pruning* (ODSAP) for protocols using server or client-based version vectors, and *ODSAP-O* for object-based version vectors. They were introduced in Piatkowski et al. (2008). These protocols are designed for environments with mobile clients, but are not yet ready for use in environments with mobile servers and clients. ODSAP* protocols constitute also a starting point for research of more advanced synchronization protocols. In the future, they can also be used as a simple reference protocols for benchmarks of new synchronization solutions.

This paper presents performance evaluation and comparison of different consistency protocols of session guarantees presented in Kobusińska et al. (2005)

when ODSAP and ODSAP-O protocols are used for server synchronization. This work shows how a system using session guarantees scales when different types of version vectors are used and the number of clients and servers changes. It is also shown, that — if needed — client-based and object-based protocols can be successfully used instead of the server-based version vectors. Presented results are based on simulation experiments; out of scope of this paper is a theoretical model of those results, which is a wide topic on its own.

2. System model

The system consists of a number of *servers* holding a full copy of a set of data objects. Clients access the data after selecting a single server and sending a direct request to the server in the form of remote procedure calls. The requests are divided into two classes: non-modifying operations (*reads*) and modifying operations (*writes*). Clients are separated from servers, and they are meant to run on other computers than servers. Clients are mobile, i.e. they can switch from one server to another. The new server selected by the client may hold replicas of objects that are inconsistent with the replicas held by the previous server. Session guarantees are expected to take care of data consistency observed by a single, migrating client.

A client can instruct the servers to check specific consistency criteria expressed using session guarantees. There are four session guarantees: Read Your Writes (RYW), Monotonic Writes (MW), Monotonic Reads (MR) and Writes Follow Reads (WFR) (Terry et al., 1994). Formal definitions of session guarantees can be found in Sobaniec (2005).

A client requesting RYW session guarantee expects that every read operation will reflect all previous writes issued by him, regardless of switching to another server during the session. This guarantee can be exemplified by a user writing a TODO list to a file. When he is traveling between different locations and servers, he wants to recall the most urgent tasks on the TODO list. Without RYW session guarantee the read may miss some recent updates of the list.

The MW session guarantee globally orders writes of a given client. Let us consider a counter object with two methods for updating its state: *increment*, and *set*. A user of the counter issues the *set* function at first, and then updates the counter by calling *increment* function. Without MW session guarantee the final result would be unpredictable, because it depends on the order of the execution of these two functions.

A client requesting MR knows that his read operation will always return a state newer or simultaneous to the one observed by a previous read operation. Usage of the MR guarantee can be illustrated by a mailbox of a traveling user. The user opens the mailbox at one location, and reads a few emails. Afterwards, he opens the same mailbox at a different location; he expects to see at least all the messages he has seen previously. The new state may not reflect the most recent one, but must be at least as new as the previously observed state.

WFR session guarantee keeps track of causal dependencies resulting from the client's operations. As an example, let us consider a discussion forum, where a user has read some posts. Some time later the user wants to post a reply to a message he has read. The new message must be submitted to a server that knows the post to which the user is going to reply. WFR session guarantee may solve the problem by tracking causal dependency between the read of the original message and the posting (write) of the reply.

It is important to note that differences between data centric and session guarantees approaches are essential. The dissimilarity is caused by a very different client cooperation model and consistency properties. This can be seen when one tries to achieve session guarantees using DSM protocols and vice versa. In Brzeziński et al. (2004) it is proved that only if we enable all possible session guarantees, we can get a causal consistency known from DSM systems. In Brzeziński et al. (2003) it is proved that enabling three of four guarantees is required to get a PRAM consistency. Thus, achieving data-centric consistency using session guarantees is impossible. Also, achievement of client-centric consistency using data-centric approach essentially requires using atomic consistency, which is very inefficient.

Let us consider a client requiring just the RYW session guarantee. The client issues a write operation at server S_1, then migrates to server S_2, and issues a read at server S_2 on the same or dependent data item. Even if the replicas are synchronized according to strong consistency model, like sequential consistency, it is still possible that the client migrates to server S_2 before the update reaches it. As a result, the client will observe a state of the new server that does not reflect modifications of his previous write, which is a violation of RYW session guarantee. This would not happen if no new client operation in the whole system is allowed to be processed until all previously issued operations are finished and replicated to all other servers, which effectively means using atomic consistency. Moreover, the client cooperation model in session guarantees differs from the one in DSM systems. In session guarantees, if a set of clients is requesting operations only from a subset of all servers, there is no need to synchronize operations among all servers, but only among the subset. This is not true in DSM, when using, for example, atomic consistency. Some of those problems are solved in voting based protocols, but they still require a synchronous update of all replicas in a write quorum (Gifford, 1979; Jajodia & Mutchler, 1987).

In session guarantees, version vectors are used to efficiently represent sets of writes resulting from session definitions. Version vectors are conceptually similar to vector clocks (Mattern, 1988; Baldoni & Raynal, 2002) used by many distributed systems. Vector clock is a logical approximation of global time. It is an extension of Lamport's logical clock (Lamport, 1978), capable of tracking causal dependencies between events in a distributed system. Vector clock is a vector of integer numbers: $V = \left[v_1 \; v_2 \; \ldots \; v_N \right]$, where N is the length of the vector. Every server S_j maintains its own vector V_{S_j}. Depending on the type of the version vector, particular positions of it are interpreted in a different manner.

In case of server-based version vectors the j-th position $V_{S_j}[j]$ is a logical scalar clock of the server S_j, while $V_{S_j}[k]$ for $k \neq j$ is a representation of the logical clock of server S_k as observed by S_j. Version vectors used by consistency protocols of session guarantees are updated during performing writes by servers. This means that the i-th position of a vector $V_{S_i}[i]$ shows how many write operations S_i has performed since the beginning of its work. Likewise, $V_{S_i}[j]$, $j \neq i$ is a number of writes performed by S_j as observed by S_i. Because only server S_i can update the i-th position of a version vector, the resulting version vectors are globally unique. It is also important to note that version vectors are monotonically increasing, as every server S_i can only increase its i-th position, and update of the j-th ($j \neq i$) position is only possible by receiving a newer vector from another server. In this case the position in the V_{S_i} vector is set to the maximum of its own vector and the new vector at position j. In this way version vectors may be used to efficiently represent growing sets of write operations, which are the basics of session guarantees operation.

In the case of client-based version vectors, values stored in the vectors are associated with clients. The length of the vector is equal to the number of clients N_C and every client should have assigned a numerical id. The i-th position $V_{S_j}[i]$ shows how many write operations issued by a client C_i are known to the server S_j. When a client C_i is requesting a write operation at the server S_j, server increases the i-th position in its vector$V_{S_j}[i]$ creating an unique timestamp of the operation as client can be connected only to one server at a time, and cannot request two write operations concurrently.

In the object-based version vectors, positions in vectors represent the number of write operations requested on individual objects. The i-th position $V_{S_j}[i]$ shows how many write operations that were performed on the i-th object are known to server S_j. Because many clients can concurrently request operations on the same object, the protocol must be extended, so that every operation could be uniquely timestamped. This can be achieved by global ordering of writes on respective objects (Sobaniec, 2005).

Further description and formal definitions of version vector representation of required write sets, session guarantees and client protocol and can be found in Terry et al. (1994) and Sobaniec (2005).

3. The synchronization protocol

The synchronization protocol proposed in Sobaniec (2005), using periodic broadcasting of updates has a few drawbacks, which do not allow for proper comparison of session guarantees protocols, especially the object-based version vector protocol. When using periodic synchronization, one must specify how often servers will exchange information. This period depends on many factors, including usage characteristics, frequency of client migration, etc. Additionally, if update propagation between servers is handled in synchronous manner, then it can be shown that the type of version vectors does not influence global per-

formance. When object-based vectors are used, the situation is even worse: because of the need of global ordering of write operations, clients must often wait for one or more synchronization periods even if no client has migrated between servers.

Another possible approach to server synchronization is to use the *anti-entropy* protocol (Petersen et al., 1997) used in the Bayou system. Unfortunately, in a mobile environment, where clients migrate often, it has similar drawbacks as the periodic synchronization protocol. Additionally, anti-entropy protocol parameters would have to be tuned, presumably independently for every protocol and its variation.

Therefore a new protocol, called *On-Demand Synchronization Algorithm with Pruning* (ODSAP), was proposed. There are two version of that protocol: the first one uses client- or server-based version vectors (ODSAP), and the second one uses object-based version vectors (ODSAP-O). In this approach no periodic updates are sent. Instead, every server that cannot process the request because of required session guarantees, sends a synchronization request message to all other servers. The client request is suspended until the server gets all operations that are needed to satisfy the required session guarantees.

In order to present ODSAP synchronization protocols, it is necessary to introduce some basic operations. *Send* is understood as a network communication primitive allowing to send a message to a remote node. `wait` causes a calling thread to suspend until the `signal` operation is executed. The function $T(op)$ returns a version vector timestamp assigned earlier to the operation op. The type of operation may be determined by the `iswrite`(op) function. *Deliver* means that a result of a remote call can be delivered to the application.

In order to be able to determine which operations can be processed while preserving session guarantees, each client C_i maintains two version vectors W_{C_i} and R_{C_i} representing the set of writes it has requested, and the set of writes observed by requested reads. Along with each request a client sends a version vector V_{C_i} describing its consistency requirements. The version vector is calculated based on version vectors W_{C_i}, R_{C_i} and the set of required session guarantees. This is presented in Algorithm 1.

Before performing a new operation a server S_j checks whether its version vector V_{S_j} dominates the version vector V_{C_i} received from client. The domination is denoted by $V_{S_j} \geq V_{C_i}$ and is fulfilled when $\forall k : V_{S_j}[k] \geq V_{C_i}[k]$. Such domination means that the required session guarantees are preserved, otherwise the server must be updated before performing the requested operation. If the operation cannot be performed without violating session guarantees, it means that there are some operations in the system seen by the client, but not performed by the server. The server is then updated by exchanging information with other servers. Every server records write operations it has performed in an ordered history, so the exchange of writes performed in the past is possible. This exchange is done using server synchronization protocol, the main subject of this paper.

Algorithm 1 Client side of ODSAP and ODSAP-O protocols

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

Upon receiving Repl $\langle op, res, V_{S_j} \rangle$ **from server** S_j **at client** C_i
9: **if** `iswrite`(op) **then**
10: $\quad W_{C_i} \leftarrow \max(W_{C_i}, V_{S_j})$
11: **else**
12: $\quad R_{C_i} \leftarrow \max(R_{C_i}, V_{S_j})$
13: **end if**
14: deliver res

The server part of the protocol is presented in Algorithm 2. When a server S_j receives a request from a client with a vector V_{C_i}, which is not dominated by server internal vector V_{S_j}, the server must delay performing the request until it executes all operations needed to preserve session guarantees. It also sends a synchronization request to other servers (line 2). The synchronization request message includes a server internal vector V_{S_j}, to indicate which operations the S_j server has already performed, and the vector V_{C_i} received from the client, to indicate which operations are required by the client, but were not yet performed by the server. Other servers respond asynchronously to S_j synchronization request only if they have operations that are needed to process the suspended operation requested by the client (lines 15-19). Server S_j gathers those responses (lines 20-27) and processes suspended requests when appropriate. Because ODSAP operation does not depend on frequency of client migrations, or any other arbitrary parameter, it can be used to compare guarantees using different types of version vectors.

As shown in Piatkowski et al. (2008) ODSAP combined with object-based vectors can result in a deadlock. To solve this issue a modification was introduced into ODSAP resulting in ODSAP-O protocol. A complete description of ODSAP and ODSAP-O can be found in Piątkowski, Sobaniec and Sobański (2010).

Algorithm 2 ODSAP — server side

Upon receiving $\mathsf{Req}\,\langle op, V_{C_i}\rangle$ **from client** C_i **at server** S_j
1: **if** $\left(V_{S_j} \not\geq V_{C_i}\right)$ **then**
2: send $\mathsf{SyncReq}\left\langle V_{S_j}\right\rangle$ to all other servers
3: **end if**
4: **while** $\left(V_{S_j} \not\geq V_{C_i}\right)$ **do**
5: `wait`
6: **end while**
7: perform op and store results in res
8: **if** `iswrite`(op) **then**
9: $V_{S_j}[j] \leftarrow V_{S_j}[j] + 1$
10: $M_{S_j}[j] \leftarrow V_{S_j}$
11: timestamp op with V_{S_j}
12: $H_{S_j} \leftarrow H_{S_j} \cup \{op\}$
13: **end if**
14: send $\mathsf{Repl}\left\langle op, res, V_{S_j}\right\rangle$ to C_i

Upon receiving $\mathsf{SyncReq}\,\langle V_{S_i}\rangle$ **from server** S_i **at server** S_j
15: $M_{S_j}[i] \leftarrow V_{S_i}$
16: $H_{\text{diff}} \leftarrow \{w_k \in H_{S_j} : V_{S_i} \not\geq T(w_k)\}$
17: **if** $H_{\text{diff}} \neq \emptyset$ **then**
18: send $\mathsf{Upd}\,\langle S_i, H_{\text{diff}}\rangle$ to S_i
19: **end if**

Upon receiving $\mathsf{Upd}\,\langle S_i, H_{\text{diff}}\rangle$ **at server** S_j
20: **foreach** $w_i \in H_{\text{diff}}$ **do**
21: **if** $V_{S_j} \not\geq T(w_i)$ **then**
22: perform w_i
23: $V_{S_j} \leftarrow \max\left(V_{S_j}, T(w_i)\right)$
24: $H_{S_j} \leftarrow H_{S_j} \cup \{w_i\}$
25: **end if**
26: **end for**
27: `signal`

On idle event at server S_j
28: $V_{\min} \leftarrow V_{S_j}$
29: **for** $k = 1 \ldots N_S, k \neq j$ **do**
30: $V_{\min} \leftarrow \min\left(V_{\min}, M_{S_j}[k]\right)$
31: **end for**
32: $H_{S_j} \leftarrow H_{S_j} \setminus \{w_i \in H_{S_j} : V_{\min} \geq T(w_i)\}$

4. Simulations

4.1. The simulation environment

Consistency protocols of session guarantees using version vectors were evaluated by means of simulations ran in the OMNeT++ environment (omnet, 2008). The tests covered different aspects of performance evaluation: response times, overall throughput, scalability, communication overhead. All tests were carried out using different input parameters: the number of servers, clients, data objects, migration intensity, update frequency etc.

Simulations were carried out in the following manner. At the beginning of a simulation every client chose randomly:

- a subset of all available objects, on which it will perform operations,
- a set of session guarantees, which it will use for the whole time of the simulation,
- a server to which it will connect at the start.

During the simulation a client can:

- send an operation request to a server,
- migrate to other server; for the simulation purposes, it is assumed that the servers are connected in a physical ring topology and it is more likely that the client will move to a nearby server than to a distant one (according to the normal distribution),
- change the set of objects on which it is performing operations.

Every experiment was described by a number of parameters. Some of their values were based on the results of preliminary simulations, other ones were chosen arbitrarily. The list of parameters and their default values is given below:

N_S number of servers (defaults to 16 servers),
N_C number of clients (defaults to 256 clients),
N_O number of objects (defaults to 64 objects),
p_o the percentage of all objects accessed by a client; the size of the subset for a single client is equal to a random value from the uniform distribution from 1 to $2p_oN_O$ (p_o defaults to 33%),
t_e delay between events — time between consecutive events generated by a single client (sending a request, migration); defaults to a random value from the exponential distribution with a mean value of 10 s,
p_m migration probability — checked at every event (defaults to 15%),
p_w write probability — ratio of write operations to read operations (defaults to 30%),
t_r read execution time — cost of performing a read operation at a server, defaults to a random value from a normal distribution (0.2 s, 0.01),
t_w write execution time — cost of performing a write operation at a server, defaults to a random value from a normal distribution (0.25 s, 0.015),
t_{hs} history processing/generation start-up cost — it is natural that the cost of processing a received synchronization message with operations from other

server is proportional to its length; also some start-up cost, independent of the message length, must be taken into account and is expressed by this parameter; the same is true for generating a synchronization reply message (defaults to 0.01 s),

t_T running time — the virtual time of simulation (defaults to 4 h).

Servers are interconnected with a 100Mb/s Ethernet backbone and clients connect to servers via a Wi-Fi connection. Transmission times of network messages were determined using another OMNeT++ simulation with INET modules used. They were measured for a campus size network without any background traffic. To validate these simulation results, a smaller experiment using a real network environment and hardware was performed.

For every set of input parameters an experiment run was performed for each type of the version vectors. The most interesting and insightful results are presented below.

4.2. Simulations results

In general, protocols using server-based, optimized server-based and client-based version vectors result in a similar performance. The reason for these similarities is the final effect of all protocols — in the long run all of them will perform all operations at all servers. The object-based protocol performs slightly differently. This is expected, as it is the only one using additional global synchronization to assure global ordering of write operations (Sobaniec, 2005).

Fig. 1 presents an average response time seen by a client when performing operation at a server, for different numbers of clients and servers. First thing to note is how the system performs without replication, i.e. when there is only one server. As can be seen, a system using replication performs considerably better than a centralized system without replication. This proves that the presented approach is reasonable.

The experiments were carried out using server- and object-based version vector protocols. Results for optimized server-based and client-based version vectors were nearly identical to the server-based version vectors. As can be seen, for a constant number of clients there exists an optimal number of servers, when the average response time experienced by clients is minimal. Adding more servers increases that average, but not drastically. Much worse is the case when the number of servers is too low. Also, one can see that there are no significant differences between performance of different session guarantees protocols.

When only one client is accessing the system, adding more servers increases response times, which is an expected result — when a client migrates, it needs to wait for servers to synchronize missing operations. On the other hand, the added time is not significant, and only seen by a client just after migration.

Histograms of the average response time in Fig. 2 present distribution for the case with 9 servers and 200 clients. Distributions of response times of the optimized server-based and client-based protocols are almost identical to the server-based protocol. This histogram explains why the total average of the response times experienced by clients is a relatively high value. Most operations are completed very quickly, but a small percentage of operations requires a very long time and strongly influences the average. Such a long response time occurs when a client migrates to a server, which has not synchronized its state with others for a long period of time. In this situation the server must process a lot of operations to ensure session guarantees.

The object-based protocol (see Fig. 2) has different distribution than other protocols. It delivers a smaller number of responses fast, but also completes a considerably smaller number of responses with a very long response time. The cause for this is a different frequency of synchronizations between servers. In object-based protocol, not only migration of clients triggers synchronization, but also execution of write operations issued on different servers, yet on the same object. In effect, one can turn more medium response times into a smaller number of a very long completion delays, which may be practical in some applications. Comparing object-based protocol to other ones, we must note that the results of the object-based solution do not include the cost of generating a globally unique sequence number in a distributed environment. So, the results are not biased by the specific distributed sequence generation protocol, but are lower than they would be in a real life scenario.

For the constant number of servers and objects and with increasing number of clients, the average response time increases, exactly as can be expected. However, if the number of servers and clients is constant, then even when the number of objects is increased, the average response time remains constant (Fig. 3). Only for a small number of objects (< 60 objects with 32 servers and 256 clients) the object-based version vectors protocol shows noticeably worse performance, which becomes really bad for just a few objects (< 4). In such case, all clients are issuing operations on the same objects and almost every write operation needs to be arranged according to the distributed sequence number generator. Such event also causes server synchronization. For a larger number of objects all protocols give approximately the same response times. Insensitivity to the number of objects is a positive aspect of all compared protocols.

Fig. 4 shows how the observed response times for different number of writes performed by clients. Write probability is the percentage of the write operations from all operations issued by the clients. As can be expected the more writes clients request the more synchronization is required between servers and average response time increases. Once more, the difference between protocols based on different version vectors is negligible, even including the object-based protocol.

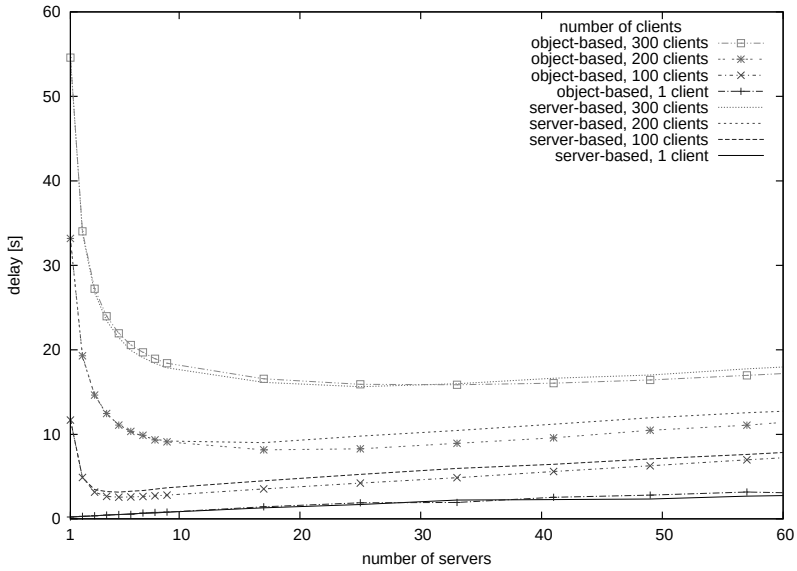

Figure 1: Response time depending on the number of servers
$N_O = 64$, $p_m = 15\%$, $p_w = 30\%$

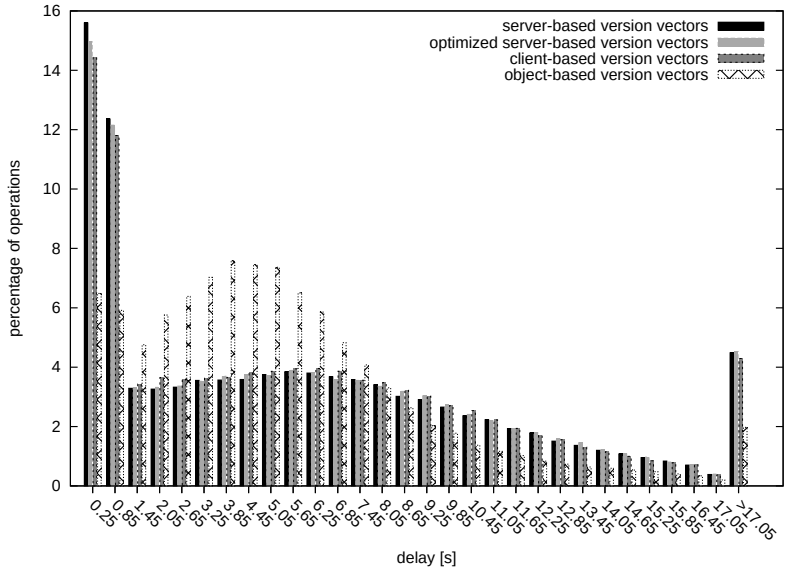

Figure 2: Histogram of response times
$N_S = 9$, $N_C = 200$, $N_O = 64$, $p_m = 15\%$, $p_w = 30\%$

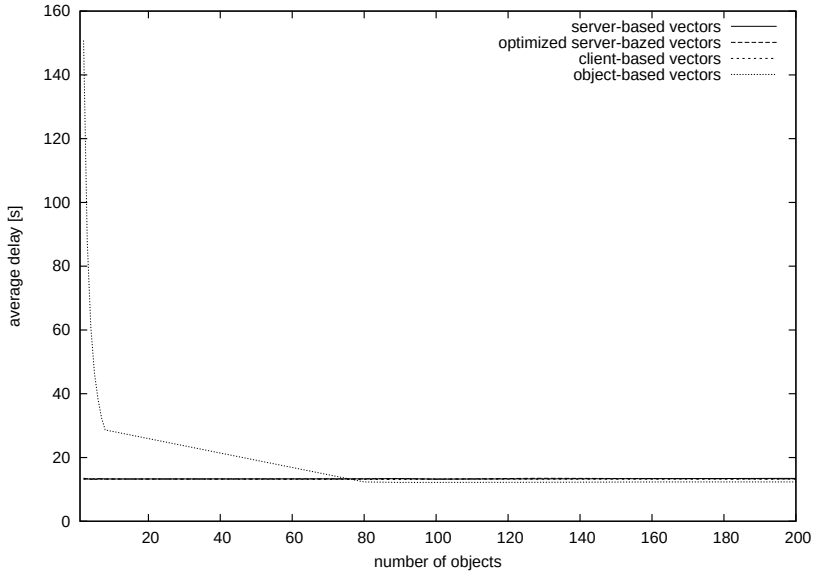

Figure 3: Response times depending on the number of objects
$N_S = 16$, $N_C = 256$, $p_m = 15\%$, $p_w = 30\%$

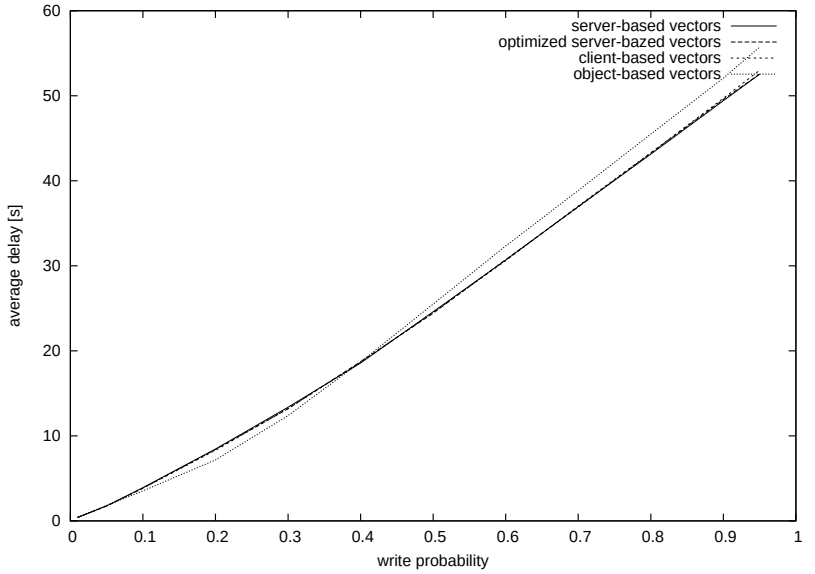

Figure 4: Response times depending on the percentage of write operations
$N_S = 16$, $N_C = 256$, $N_O = 64$, $p_m = 15\%$

Figs. 5, 6 and 7 present another aspect of efficiency of compared protocols. These graphs show the average number of messages sent through the network per client operation performed in the system. All synchronization request related messages are taken into account. As before, all protocols behave similarly, with the exception of the object-based protocol, which requires more synchronizations between servers. The graphs do not include messages needed by the object-based protocol to select a value of the distributed sequence number for writes scheduling. For a larger number of objects the object-based protocol needs much less messages, because in this case one object is rarely accessed by more than one client at a time. However, when the number of clients accessing a set of objects grows, the number of messages sent becomes higher compared to other protocols. Fig. 7 shows also that the number of messages being sent per request (except for the object-based protocol) does not depend on the number of objects in the system.

Performance and scalability of the whole system is presented in Fig. 8. It shows that the system scales well and that for every client number up to several hundred, a few dozen servers are able to process all issued requests, without queuing them due to lack of resources.

Fig. 9 shows how big is the processing capacity of servers and how the system scales in terms of percentage of time servers were busy processing requests. It is natural that a given number of servers is only capable of handling some maximum number of clients. For a greater number of clients servers will saturate and will not be able to perform more requests. This state of saturation is represented by the black zone on the graph. As presented in the graph, the number of clients causing saturation of servers grows only logarithmically in function of the number of servers. It presents the limits of the scalability for the given set of system parameters (i.e. processing times of operations).

Comparing the point of saturation with a throughput of a system shown in Fig. 8, it is worth noting that the point of saturation is the same point at which system has the maximal throughput, for a given number of servers. If the number of clients is kept constant and the number of servers is increased, then workload and throughput drop. The reason for that is an increased cost of synchronization between greater number of servers. If the number of clients is increased, then throughput does not increase (and can even decrease slightly), because servers are becoming overloaded. This shows that to get better performance, additional ODSAP servers should be added to the system only when old servers are running near the maximum of possible workload.

5. Conclusions and future work

In this paper we have evaluated protocols that can be used for ensuring session guarantees using ODSAP protocols for server synchronization. In the original specification of session guarantees a server-based version vectors were used. We have shown that protocols using client-based and object-based version vectors

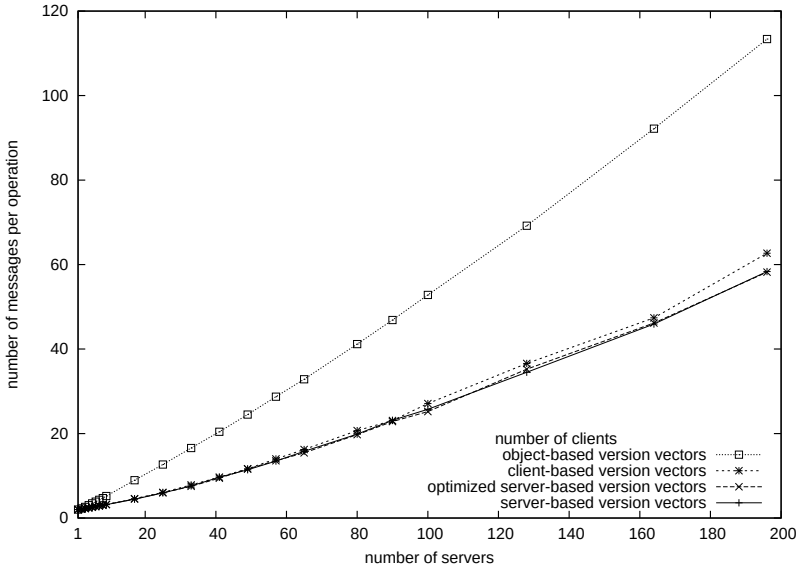

Figure 5: Messages per request, depending on the number of servers
$N_C = 256$, $N_O = 64$, $p_m = 15\%$, $p_w = 30\%$

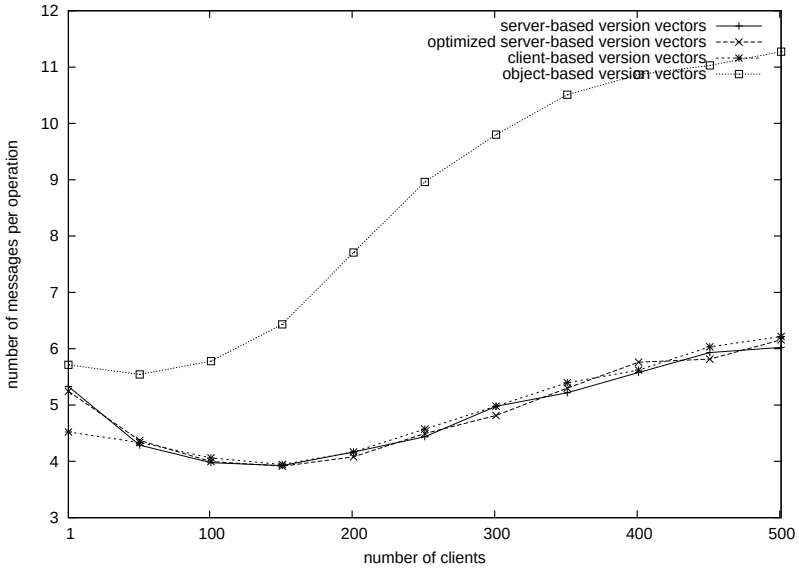

Figure 6: Messages per request, depending on the number of clients
$N_S = 16$, $N_O = 64$, $p_m = 15\%$, $p_w = 30\%$

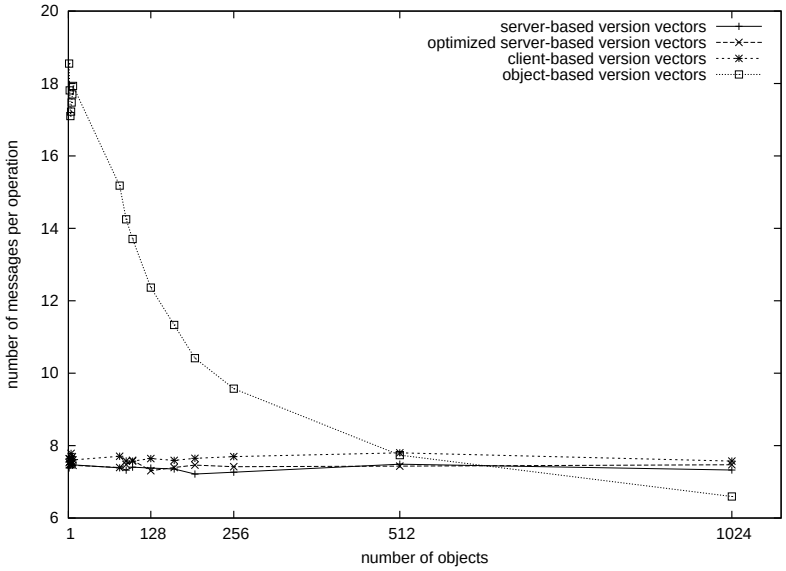

Figure 7: Messages per request, depending on the number of objects
$N_S = 16$, $N_C = 256$, $p_m = 15\%$, $p_w = 30\%$

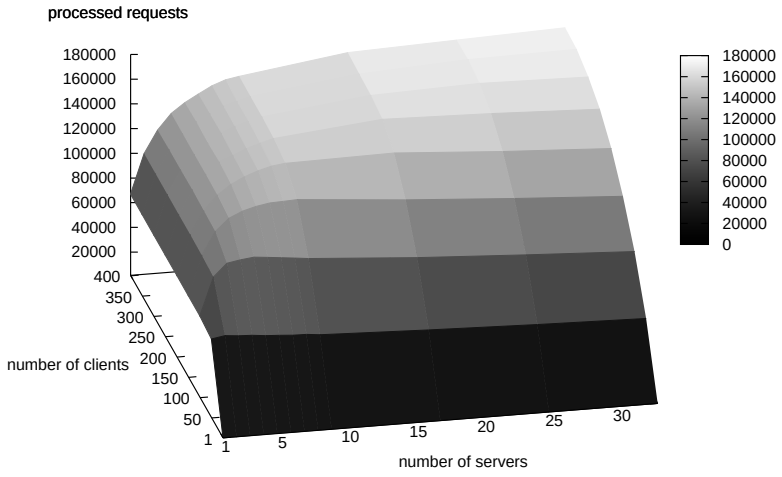

Figure 8: Throughput
$N_O = 64$, $p_m = 15\%$, $p_w = 30\%$

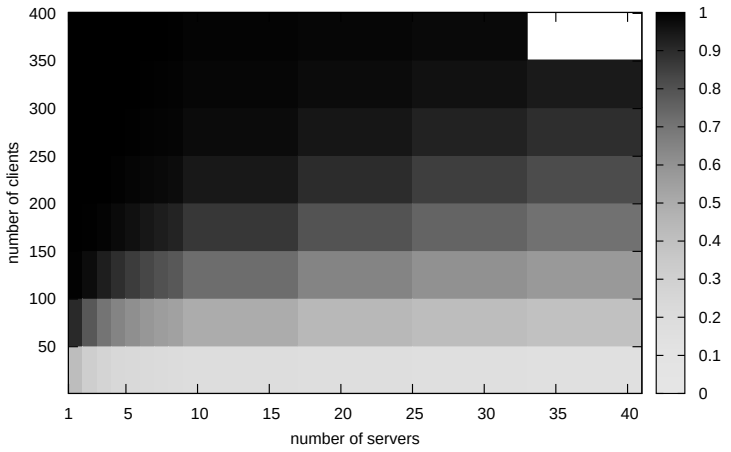

Figure 9: Workload
$N_O = 64$, $p_m = 15\%$, $p_w = 30\%$

are similar in performance to the server-based solution. It is possible to choose a proper protocol for specific needs of the application.

From the presented evaluation one can also deduce basic properties of system scalability. Most importantly, the results show that using proposed session guarantees and the synchronization protocols offers considerably better performance than a scenario with no replication. Also, the fact that average response time seen by a client is constant for varying number of objects and for all protocols except object-based is promising for practical implementations. The system load was confirmed to scale well as the number of clients grows. Obtained performance data can be used in the future to compare the performance of new synchronization protocols to ODSAP.

Further work is needed on synchronization protocols to support more general environments. During the evaluation of different types of version vectors we have observed that the periodic synchronization (and presumably, for the same reasons, the basic anti-entropy algorithm) is not suitable for frequently migrating clients. On the other hand, ODSAP protocols tend to perform poorly when migration of clients is an uncommon event. Developing a synchronization protocol that combines advantages of both solutions is now an interesting and useful challenge.

Moreover, work has been started towards a fully mobile environment, where not only clients, but also servers, are mobile and the network connecting servers may fail or became temporarily unavailable due to server movement and network partitioning.

References

Baldoni, R. and Raynal, M. (2002) Fundamentals of distributed computing: A practical tour of vector clock systems. *IEEE Distributed Systems Online* **3**(2).

Brzeziński, J., Kalewski, M. and Sobaniec, C. (2007) Safety of a session guarantees protocol using plausible clocks. In: *Proc. of the Seventh Int. Conf. on Parallel Processing and Applied Mathematics (PPAM'2007).* **LNCS 4967**, 1–10.

Brzeziński, J., Sobaniec, C., and Wawrzyniak, D. (2003) Session guarantees to achieve PRAM consistency of replicated shared objects. In: *Proc. of the Fifth Int. Conf. on Parallel Processing and Applied Mathematics (PPAM'2003).* **LNCS 3019**, 1–8.

Brzeziński, J., Sobaniec, C. and Wawrzyniak, D. (2004) From session causality to causal consistency. In: *Proc. of the 12th Euromicro Conf. on Parallel, Distributed and Network-Based Processing (PDP2004).* 152–158.

Fidge, C. (1991) Logical time in distributed computing systems. *Computer* **24**(8), 28–33.

Gifford, D. (1979) Weighted voting for replicated data. In: *Proc. of the 7th ACM Symp. on Operating Systems Principles (SOSP).* ACM Press, 150–162.

Jajodia, S. and Mutchler, D. (1987) Dynamic voting. *SIGMOD Rec.* **16**(3), 227–238.

Kobusińska, A., Libuda, M., Sobaniec, C. and Wawrzyniak, D. (2005) Version vector protocols implementing session guarantees. In: *Proc. of Int. Symp. on Cluster Computing and the Grid (CCGrid 2005).* Springer, 929–936.

Lamport, L. (1978) Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* **21**(7), 558–565.

Mattern, F. (1988) Virtual time and global states of distributed systems. In: Cosnard et al., eds., *Proc. of the Int. Conf. on Parallel and Distributed Algorithms*, Elsevier Science Publishers B.V., 215–226.

omnet (2008) Omnet++ discrete event simulation system. `http://www.omnetpp.org/`.

Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M. and Demers, A.J. (1997) Flexible update propagation for weakly consistent replication. In: *Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP-16).* ACM Press, 288–301.

Piatkowski, L., Sobaniec, C. and Sobanski, G. (2008) On-demand server synchronization algorithms for session guarantees. In: *Proc. of the 23rd International Symposium on Computer and Information Sciences (ISCIS 2008).* IEEE Computer Society, 1–4.

Piatkowski, L., Sobaniec, C. and Sobanski, G. (2010) On-Demand Server Synchronization Protocols of Session Guarantees. *Foundations of Computing and Decision Sciences*, **35** (4), 307–324.

Ratner, D., Reiher, P., and Popek, G. (1997) *Dynamic Version Vector Maintenance*. Technical Report CSD-970022, Univ. of California, Los Angeles.

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

Tanenbaum, A.S. and van Steen, M. (2002) *Distributed Systems — Principles and Paradigms*. Prentice Hall, New Jersey.

Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M., Theimer, M. and Welch, B.W. (1994) Session guarantees for weakly consistent replicated data. In: *Proc. of the 3rd Int. Conf. on Parallel and Distributed Information Systems (PDIS 94)*, IEEE Computer Society, 140–149.

Torres-Rojas, F.J. and Ahamad, M. (1999) Plausible clocks: Constant size logical clocks for distributed systems. *Distributed Computing*, **12**(4), 179–196.