Performance Evaluation of Consistency Protocols of Session Guarantees*

Łukasz Piątkowski, Cezary Sobaniec, and Grzegorz Sobański

Institute of Computing Science
Poznań University of Technology, Poland
{Lukasz.Piatkowski,Cezary.Sobaniec,Grzegorz.Sobanski}@cs.put.poznan.pl

Abstract. Session guarantees define consistency properties of replicas in a distributed system from a point of view of a single, migrating client. Consistency protocols of session guarantees are used for efficient representation of sets of writes performed in the system, which are required to achieve consistency. This paper presents performance evaluation of consistency protocols of session guarantees using different forms of write sets representation. The evaluation was carried out by means of simulation experiments.

1 Introduction

For many years distributed systems have been gaining more and more popularity and have become a standard way of delivering services. With a decreasing cost of mobile devices like laptops, PDAs, smartphones and a 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 focus in fields of sensor and mobile ad-hoc networks. In these environments 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 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 is a problem of data consistency, which arises when replicas are modified. In a mobile environment clients are free to change their location and the server at which they access data. Intuitively, the client wants to continue processing even if it has switched servers, with all of its previous operations done and theirs results available.

Numerous consistency models were 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. Unfortunately, these models are not well suited for mobile and wireless networks. On the one hand,

* 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.

weak DSM models which are possible in such environments are hard to use by the user, while strong models are easy to use, but very hard, or even impossible, to achieve in a highly mobile and asynchronous system [2]. On the other hand, DSM models maintain consistency from a point of view of a data object, independently of a number of clients and their locations.

Another approach, designed with mobile environments in mind, are *Session guarantees* [3], also called *client-centric* consistency models [1]. They have been proposed to define required properties of consistency from a point of view of a single migrating client. In general: in system based on session guarantees consistency is always expressed as a set of write operations which are known to the client. This set of writes is growing monotonically as the client is requesting new operations to be performed by one of the servers. Meanwhile other clients may perform some operations at the same server, so the client also learns new write operations issued by other clients.

Protocols implementing session guarantees must efficiently represent sets of operations performed in the system. Version vectors based on vector clocks [4, 5] may be used for this purpose. The original protocol presented in [3] used server-based version vectors, where each position of the version vector represents the number of writes performed by a given server. [6] shows that other approaches are also possible: protocols using client-based or object-based version vectors. Positions in the vector representing respectively: the number of writes performed by the clients or the number of writes performed on objects. In different applications different version vectors may be preferred, especially when considering some variants of version vectors like dynamic sized vectors [7] or plausible clocks [8].

One of the key elements of session guarantees protocols is a server synchronisation protocol. For early performance evaluation a very simple, periodic server synchronisation method was used [9]. To overcome its limitations, new synchronisation protocols were proposed: *ODSAP* for protocols using server or client-based version vectors, and *ODSAP-O* for object-based version vectors. They are introduced in [10]. These protocols are designed for mobile client environment, but are not yet ready for use in a environment with mobile servers and clients. ODSAP protocols are also a starting point for a research of more advanced synchronisation protocols. In the future, they can also be used as a simple reference protocol for benchmarks of new synchronisation protocols.

This paper presents performance evaluation and comparison of different consistency protocols of session guarantees presented in [6] when ODSAP and ODSAP-O protocols are used for server synchronisation. This work shows how the system with session guarantees scales when different vectors are used and the client to server ratio 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.

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 can 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 servers to check specific consistency criteria expressed using session guarantees. There are four session guarantee types: *Read Your Writes* (RYW), *Monotonic Writes* (MW), *Writes Follow Reads* (WFR) and *Monotonic Reads* (MR). A client requesting RYW expects, that every read operation will reflect all of its previous writes, regardless of switching meanwhile to another server. The MW session guarantee orders globally writes of a given client. A client requesting MR knows, that his read operation will always return a state newer or equal to the one observed by a previous read operation. WFR session guarantee keeps track of causal dependencies resulting from the client's operations. Formal definitions of all guarantees can be found in [9].

It is important to note, that differences between data centric and session guarantees approaches are essential and clearly seen when we try to compare them. 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 [11] it is proved, that only if we enable all possible session guarantees, we can get a causal consistency known from DSM systems, and in [12] that enabling three of four guarantees is required to get a PRAM consistency. Thus trying to implement DSM consistency using session guarantees is inefficient. Also trying to get session guarantee consistency using DSM algorithms is very ineffective. Let us consider a client with just MW guarantee enabled issuing two write operations w_1 and w_2 at two different servers S_1 and S_2, respectively. If a strong consistency of Sequential Order is enabled on the server side, it is still possible, that during the synchronisation of servers, the final order of the writes will become w_2, w_1. This satisfies sequential consistency, but nevertheless violates MW guarantee. Therefore, to achieve MW guarantee, no new client's operation in the whole system may be processed until a previously issued operation is finished and replicated to all servers — and this requires 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 synchronise operations among all servers, but only among the subset. This is not true in DSM, i.e. when using Atomic Consistency, many protocols require all replicas to be updated before client's operation is finished. Some of those problems are solved in voting base protocols, but still they require a synchronous update of all replicas in a write quorum [13, 14].

In session guarantees, version vectors are used to effectively represent sets of writes resulting from session definitions. Each client maintains two version vectors representing the set of writes it has requested, and the set of writes observed by requested reads. Each server maintains a version vector updated on every write. Version vector representations of sets of writes provide sufficient conditions for assuring session guarantees. Every client along with every request sends a version vector representing the set of writes that are expected to be performed by the destination server before proceeding to the current operation. The server can check if session guarantees are preserved by comparing the

received vector with its own. If the operation cannot be performed without violating session guarantees, it means that there are some operations in the system seen by the client, bu not by the server. The server is then updated by synchronising with other servers. Formal definitions and exact algorithms can be found in [3, 9].

Every server records write operations it has performed in an ordered history. Servers exchange information about writes performed in the past in order to synchronise the state of replicas. This synchronisation procedure eventually causes total propagation of all writes directly submitted by clients.

3 The Synchronisation Protocol

The periodic synchronisation protocol, that was used at first, 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 synchronisation, one must specify how often servers will exchange information. This period depends on many factors, including usage characteristics, frequency of clients' migration, etc. Additionally, if write sets exchange between servers is synchronous, then it can be shown that the type of version vectors used would not matter for the global performance. When using object-based vectors the situation is even worse: because of the need of global ordering of write operations, clients must often wait for one or more synchronisation periods even if no client has migrated.

Another possible approach to server synchronisation is to use the *anti-entropy* protocol [15] used in the Bayou system. Unfortunately, in a mobile environment, where clients migrate often, it has similar drawbacks as the periodic synchronisation protocol. Additionally, anti-entropy protocol parameters would have to be tuned, presumably independently for every protocol and protocol variation.

Therefore new *On-demand Synchronisation Algorithm with Pruning* was proposed. There are two version of that protocol: one for client and server version vectors (*ODSAP*) and second for object vectors (*ODSAP-O*). In this approach no periodic updates are sent. Instead, for every client's request coming to a server S_j, a write set requirement, represented as a vector in the request, is checked. If the server cannot process the request because of enabled session guarantees, it sends a synchronisation request message to all other servers. The client request is hold back until the server gets all operations that are needed to satisfy the given session guarantees.

The synchronisation request message includes a server's internal vector V_{S_j}, to indicate which operations the server already has, and a vector W, received from a client, to indicate which operations are known to the client, but not to the server. Other servers asynchronously respond the server S_j if they have operations that are needed to process the held back operation. The S_j server gathers those responses and processes the held operations when appropriate. Because ODSAP operation does not depend on a frequency of clients migrations, or any other arbitrary parameter, it is well suited to the comparison of different types of version vectors used to represent the set of required operations. A complete description of ODSAP protocols can be found in [10].

4 Simulations

4.1 The simulation environment

Version vector protocols and system scalability were evaluated through simulation experiments ran in the OMNeT++ environment [16].

Simulations were carried out in the following manner. At the beginning of a simulation every client chooses 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 the servers are connected in a logical ring topology and it is more likely, that the client will move to a nearby server then to a distant one (a 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, others were chosen arbitrary. A list of parameters and their default values is given below:

number of servers – defaults to 16 servers,
number of clients – defaults to 256 clients,
number of objects – defaults to 64 objects,
mean client object set size – a size of a client's subset of objects, on which it performs operations, given as a percentage of the whole object set; the size of the subset will vary from 1 to $2 * mean\,client\,object\,set\,size * number\,of\,objects$; defaults to 33%,
delay between events – a time between consecutive events generated by a client (request sending, migration); defaults to a random value from the exponential distribution with a mean value of 10s,
migration probability – checked every *delay between events*; defaults to 15%,
write probability – a ratio of write operations to read operations; defaults to 30%,
read delay – a cost of performing a read operation at a server, defaults to a random value from a normal distribution (0.2s, 0.01),
write delay – a cost of performing a write operation at a server, defaults to a random value from a normal distribution (0.25s, 0.015),
history processing/generation startup cost – it is natural, that a cost of processing a received synchronisation message with operations from other server is proportional to its length; also some startup cost, independent of the message length, must be taken into account and is expressed by this parameter; the same is true for generating a synchronisation reply message; defaults to 0.01s,
running time – span of a virtual time of simulation, defaults to 4h.

Servers are interconnected with a 100Mb/s Ethernet backbone and clients connect to servers via a Wi-Fi connection. Travel times for messages are based on a simulations carried out in OMNeT++ with INET modules for a campus size network without any background traffic. To validate these results a smaller experiment was done in a real network environment.

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

4.2 Simulations results

In general, protocols using a server-based, an optimized server-based and a client-based version vectors results in a similar performance. The reason for this similarities is the final effect of all protocols – in the long run all of them will lead to performing all operations at all servers. The object-based protocol performs slightly different. This is expected, as it is the only one using additional global synchronisation to assure global ordering of write operations [9]. Results showing replication performance for a constant number of clients can be seen in Fig. 1. It presents an average delay seen by a client when performing operation at a server. The most important property to note is the initial drop of a delay as the number of servers increases above one.

Figure 1 shows also how the system behaves when the number of servers and clients changes. Especially, one can see how the system performs without replication, i.e. when there is only one server. Experiments shown were carried out using server and object version vectors protocols. Results for an 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 delay noticed by clients is minimal. Adding more servers increases that average, but not drastically. Much worse is the case when a number of servers is too low. As can be seen, system with session guarantees enabled performs considerably better than a centralized system without replication. Also, one can see that there are no significant differences between performance of different session guarantees protocols.

Histograms of the average delay in Fig. 2 present distribution of the values of delays for the case with 9 servers and 200 clients. Distributions of delays of the optimized server-based and the client-based protocols are almost identical to the server-based protocol. This histogram explains why a total average of the delays 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 that has not synchronised 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 (pattern filled in Fig. 2) has different distribution than other protocols. It delivers a smaller number of responses quite fast, but also a much smaller number of responses with a very long delay. The reason for that is the additional sequence number generation for write requests. In the object-based protocol, the cost of more medium delays can be turned into a smaller number of very long delays, which may be practical in some applications.

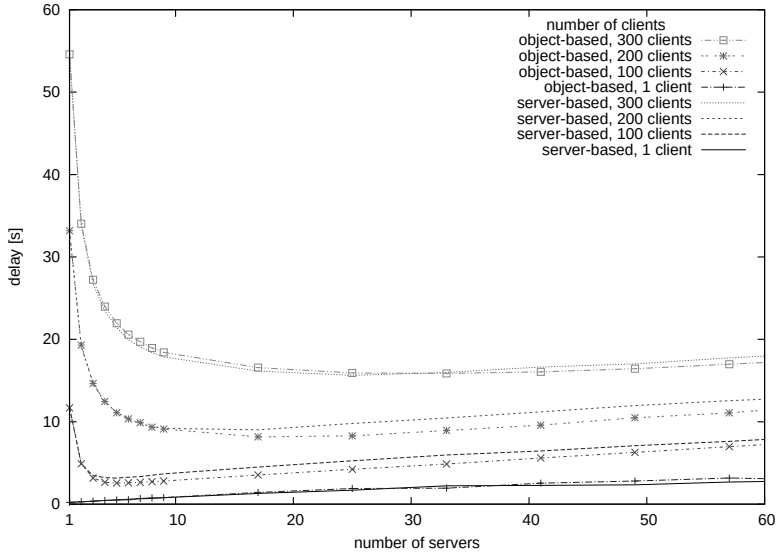

Fig. 1: Delays

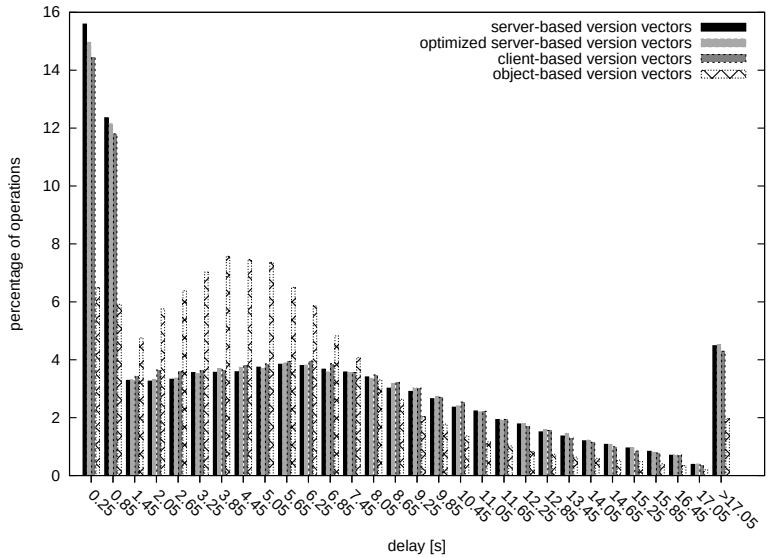

Fig. 2: Histogram of delays

For the constant number of servers and objects with increasing number of clients, the average delay 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 delay remains constant (graph not included). Only for a small number of objects (< 60 objects with 32 servers and 256 clients) the object-based version vectors protocol behaves noticeably worse, and really bad for just a few objects (< 4). This is again because of the additional sequence number generation in this protocol. For larger number of objects all protocols give the same delay. Insensitivity to the number of objects is a positive aspect of all compared protocols.

Figures 3, 4 and 5 present another aspect of the efficiency of compared protocols. These graphs show an average number of messages sent through the network per operation. All synchronisation requests messages are included. As before, all protocols behave similarly, with the exception of the object-based protocol, which requires more synchronisations between servers when different clients modify the same object. The graphs do not include messages needed by object-based protocol to select a 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 then one client at a time. However, when the number of clients accessing a set of objects grows, the number of messages sent is considerably higher comparing to other protocols. Figure 5 shows also, that number of messages being sent per request (except object-based protocol) does not depend on the number of objects in the system.

A performance and scalability of the system from its owner point of view is presented in Fig. 6. It shows that the system scales well and for every client number up to several hundreds, a few dozens of servers are able to process all issued requests, without queuing them due to lack of resources.

Figure 7 shows how big is the processing capacity of servers and how the system scales. This graph is presented as a percentage of time servers were busy processing requests. It is natural, that a given number of servers is only capable of handling some maximal 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 a black zone on the graph. As is 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 system parameters (processing times of operations).

Comparing the point of saturation with a throughput of a system shown in Fig. 6, 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 number of servers is increased, then workload and throughput drops. The reason for that is an increased cost of synchronisation between servers. If the number of clients is increased, then throughput does not increase (and can even decrease slightly), because servers are already working at their highest capacity.

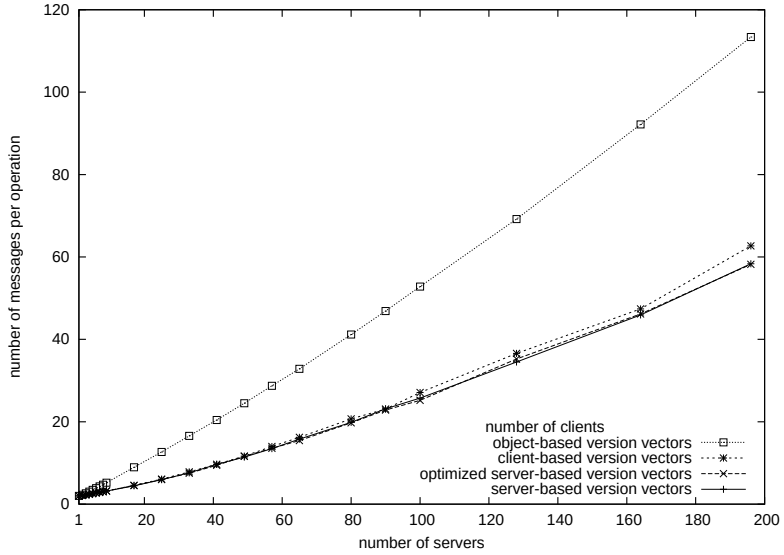

Fig. 3: Messages per request, depending on the number of servers

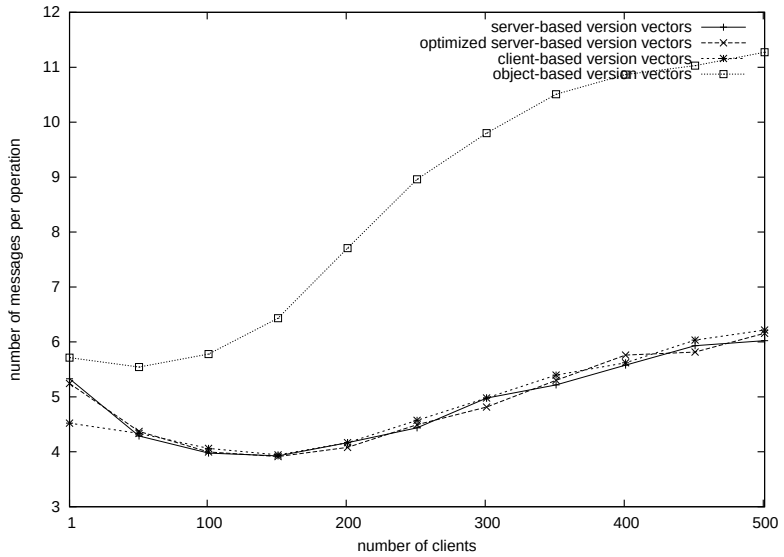

Fig. 4: Messages per request, depending on the number of clients

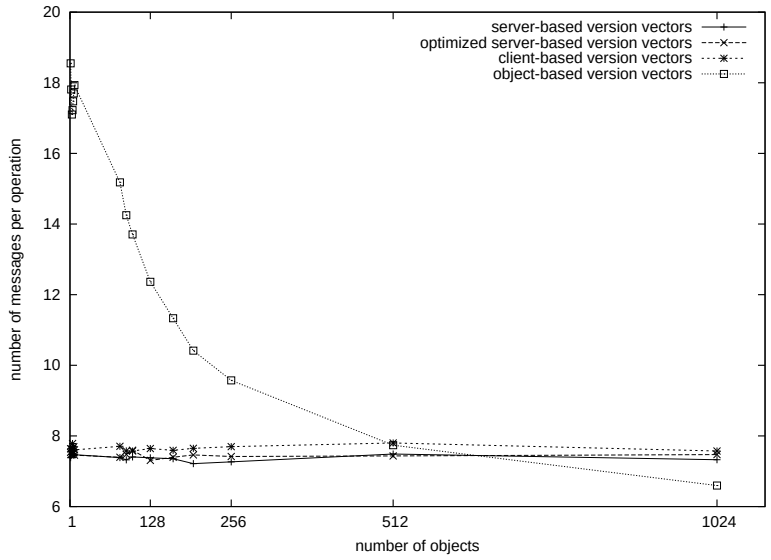

Fig. 5: Messages per request, depending on the number of objects

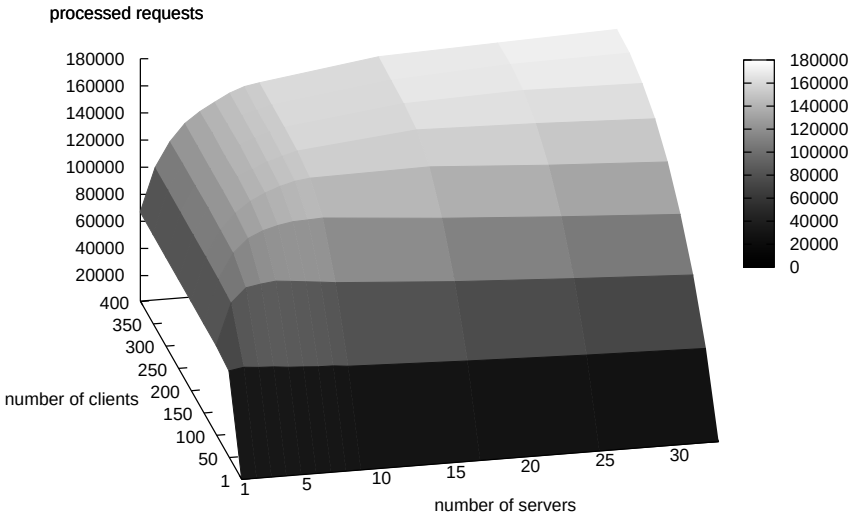

Fig. 6: Throughput

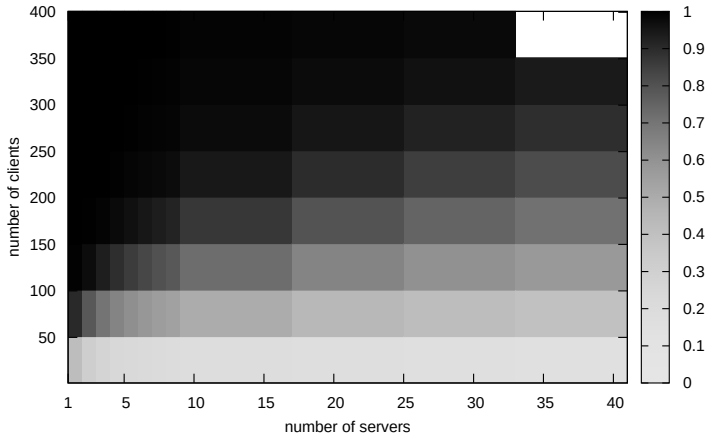

Fig. 7: Workload

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

From the presented evaluation one can also see basic properties of the system's scalability. Most importantly, the results show, that using proposed session guarantees and the synchronisation protocols offers considerably better performance than a scenario with no replication. Also, the fact that average delay 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 synchronisation protocols with ODSAP.

Further work is needed on synchronisation protocols to support more general environments. During the evaluation of different types of version vectors we have observed that the periodic synchronisation (and presumably, for the same reasons, the base 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 synchronisation protocol that supports both situations properly with session guarantees is now an important challenge.

Moreover, work is needed towards a fully mobile environment, where not only clients are mobile and network connecting servers may fail or became temporarily unavailable due to servers' movement and network partitioning.

ODSAP synchronisation protocol requires formal proofs of its correctness. Such proofs were already conducted, but length limitations of this paper does not allow us to present them. Nevertheless, these proofs were described in a separate paper and submitted for review to the same conference.

References

[1] A. S. Tanenbaum and M. van Steen, *Distributed Systems — Principles and Paradigms*. New Jersey: Prentice Hall, 2002.

[2] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[3] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. W. Welch, "Session guarantees for weakly consistent replicated data," in *Proc. of the Third Int. Conf. on Parallel and Distributed Information Systems (PDIS 94)*, (Austin, USA), pp. 140–149, IEEE Computer Society, Sept. 1994.

[4] F. Mattern, "Virtual time and global states of distributed systems," in *Proc. of the Int. Conf. on Parallel and Distributed Algorithms* (Cosnard, Quinton, Raynal, and Robert, eds.), pp. 215–226, Elsevier Science Publishers B. V., Oct. 1988.

[5] C. Fidge, "Logical time in distributed computing systems," *Computer*, vol. 24, pp. 28–33, Aug. 1991.

[6] A. Kobusińska, M. Libuda, C. Sobaniec, and D. Wawrzyniak, "Version vector protocols implementing session guarantees," in *Proc. of Int. Symp. on Cluster Computing and the Grid (CCGrid 2005)*, (Cardiff, UK), pp. 929–936, May 2005.

[7] D. Ratner, P. Reiher, and G. Popek, "Dynamic version vector maintenance," Tech. Rep. CSD-970022, Univ. of California, Los Angeles, June 1997.

[8] J. Brzeziński, M. Kalewski, and C. Sobaniec, "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*, (Gdańsk, Poland), pp. 1–10, Sept. 2007.

[9] C. Sobaniec, *Consistency Protocols of Session Guarantees in Distributed Mobile Systems*. PhD thesis, Poznań University of Technology, Poznań, Sept. 2005.

[10] L. Piatkowski, C. Sobaniec, and G. Sobanski, "On-demand server synchronization algorithms for session guarantees," in *Proc. of the 23rd International Symposium on Computer and Information Sciences (ISCIS 2008)*, (Istanbul, Turkey), pp. 1–4, Oct. 2008.

[11] J. Brzeziński, C. Sobaniec, and D. Wawrzyniak, "From session causality to causal consistency," in *Proc. of the 12th Euromicro Conf. on Parallel, Distributed and Network-Based Processing (PDP2004)*, (A Coruña, Spain), pp. 152–158, Feb. 2004.

[12] J. Brzeziński, C. Sobaniec, and D. Wawrzyniak, "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*, (Częstochowa, Poland), pp. 1–8, Sept. 2003.

[13] D. Gifford, "Weighted voting for replicated data," in *Proc. of the 7th ACM Symp. on Operating Systems Principles (SOSP)*, (Pacific Grove, USA), pp. 150–162, Dec. 1979.

[14] S. Jajodia and D. Mutchler, "Dynamic voting," *SIGMOD Rec.*, vol. 16, no. 3, pp. 227–238, 1987.

[15] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers, "Flexible update propagation for weakly consistent replication," in *Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP-16)*, (Saint Malo, France), pp. 288–301, Oct. 1997.

[16] "Omnet++ discrete event simulation system." http://www.omnetpp.org/.