REPLICATION IN SERVICE ORIENTED ARCHITECTURES

Michael Ameling, Marcus Roy
SAP Research CEC Dresden,Chemnitzer Str. 48, Dresden, Germany

Bettina Kemme
School of Computer Science, McGill University, 3480 University Street, Montreal, Canada

Keywords: Application server, replication, Web Service, simulation.

Abstract: Multi-tier architectures have become the main building block in service-oriented architecture solutions with stringent requirements on performance and reliability. Replicating the reusable software components of the business logic and the application dependent state of business data is a promising means to provide fast local access and high availability. However, while replication of databases is a well explored area and the implications of replica maintenance are well understood, this is not the case for data replication in application servers where entire business objects are replicated, Web Service interfaces are provided, main memory access is much more prevalent, and which have a database server as a backend tier. In this paper, we introduce possible replication architectures for multi-tier architectures, and identify the parameters influencing the performance. We present a simulation prototype that is suitable to integrate and compare several replication solutions. We describe in detail one solution that seems to be the most promising in a wide-area setting.

1 INTRODUCTION

Multi-tier architectures are the main building block of modern business applications. Clients submit requests to application server tier which implements the application logic (e.g., purchase orders, bookings, etc.). The persistent data is stored in a backend tier. As businesses open their applications to a wide range of clients the infrastructure is put on stringent requirements. Firstly, it has to follow a service-oriented architecture (SOA) where the business logic is exposed as services through well defined interfaces. Secondly, the platform must be able to support an increasingly heavy and complex load. Thirdly, it has to provide a certain QoS level not only to local clients but to clients across the globe. Finally, the platform must be available around the clock. One of the main techniques to achieve these objectives is replication. It allows the system to distribute the load, to provide fast access to close-by replicas and to increase availability as clients can connect to any of the available replicas. A main challenge of replication in data intensive environments is replica control, that is, the task to keep the data copies consistent.

Replication has been widely used in database systems and the implications of using replication are well understood (Gray et al., 1996; Pedone et al., 2003; Lin et al., 2005; Plattner et al., 2008). Similarly, object-based systems have explored replication as a tool to either increase fault-tolerance (Narasimhan et al., 2001; Killijian and Fabre, 2000; Frølund and Guerraoui, 2002) or improve performance (Othman et al., 2001). However, only recently replicating the middle-tier of a complex multi-tier system has been considered (Felber and Narasimhan, 2002; Narasimhan et al., 2001; Barga et al., 2002; Wu and Kemme, 2005; Salas et al., 2006; Perez-Sorrosal et al., 2007). So far, the proposed solutions have been developed for very specific scenarios, and it has become clear that no single solution will fit all. The reason is that the impact of replication on performance depends on many different parameters such as the required degree of consistency, the distribution within LANs or WANs, the workload, etc. As implementing replica control into a complex multi-tier architecture is a non-trivial task, one has to understand the implications of the different replication alternatives in order to choose the best solution for the given scenario.

In this paper we present a step by step a framework that allows reasoning about replication alternatives. We first discuss the many different ways applications

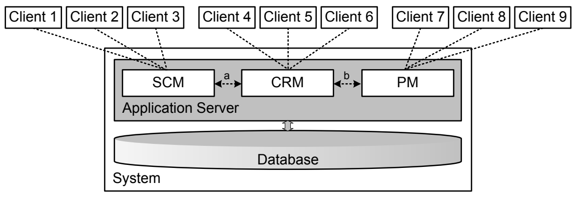


Figure 1: Multi-tier Architecture.

are deployed within a SOA, and how replication can be deployed in these environments (Section 2). Secondly, we discuss a range of replication solutions developed within the database community, and discuss their suitability for SOA (Section 3). Thirdly, we have developed a simulation framework that captures the most significant characteristics of SOA (Section 4). Thus, it serves as a first analysis platform and provides a means to compare different replication alternatives before any real, and thus complex and cumbersome implementation is done. We show an initial evaluation using our simulator (Section5).

2 REPLICATION IN SOA

2.1 Basic Architecture

A business application maintains a set of *business objects*, and defines a set of *services* as its interface. Clients can call these services to create, delete, access and manipulate the business objects. In order to provide interoperability and support heterogeneous environments, the interfaces usually follow the web-service standard. Examples of applications are "customer relationship management (CRM)", "supply chain management (SCM)" or "project management (PM)". A typical example of a service within a CRM is "customer data management" providing a set of methods that allow the customer to change customer related information such as the address. A business object in this context would assemble all information related to the customer.

Figure 1 shows the architecture in which these applications are embedded. An *application server (AS)* is a software engine hosting one or more applications. It provides the computing environment to control and schedule service executions on the applications. The applications hosted on an AS can belong to the same or different companies (e.g., in the case of data centers). In any case, the usage of AS allows for an efficient sharing of resources. In standard computing environments, an AS can host up to 20 different applications each serving up to 100 clients concurrently.

AS and backend tier, typically a database system (DBS) are usually located close to each other, often even on the same machine. AS and DBS share the responsibility of maintaining the state of the business objects. The DBS provides the persistence of business objects. However, during service execution, the AS loads the objects from the DBS into its main memory cache. While the database maintains the data in form of records of tables, the AS provides an object-oriented view. Often, a business object is assembled from many different database records providing a more appropriate abstraction for application programming. Changes on business objects are written back to the database when service execution completes.

Most commonly, each application has its own distinct business objects, and maintains its own database (or at least its own tables) within the DBS. This allows for a clear separation of components and is necessary to guarantee reusability. In order to exchange information between the applications, they have to call each others' services through the standard interfaces making one application the client of the other.

2.2 Replication

Replication can boost performance in two ways. In *cluster replication*, server replicas are used for scalability. A load-balancer component distributes incoming service requests to the replicas. As more replicas are added, the system is able to handle more load. Alternatively, replicas can be located at distant geographic locations. Thus, clients can connect to the closest replica, avoiding high-latency WAN communication between client and server. Also, the system remains available even if remote servers (e.g., head quarters) are currently not accessible.

As SOA environments maintain a considerable amount of data, we cannot simply replicate server instances – we also have to replicate data. As data can change, this requires some form of *replica control* whose task it is to keep the data copies consistent. Since the maintenance of data is distributed across AS and DBS the question arises who is responsible for replica control. There exists a large body of database replication solutions, making it attractive to rely on replica control at the DBS level. However, this does not seem practical in SOA because it either affects efficiency or consistency. The business objects maintained by the AS layer can be viewed as a cache of the data stored at the DBS. If only the DBS layer is responsible for keeping data copies consistent, each access to a business object has to be redirected to the DBS in order to guarantee to read the latest version of the data or to make sure that all copies are updated. With this, the efficiency advantages of the AS object

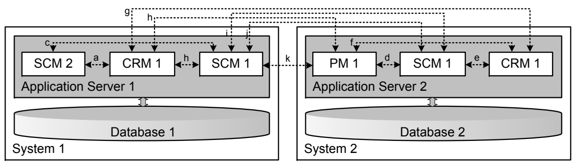


Figure 2: Replication in SOA.

cache are lost. Alternatively, one could choose to use the cache anyways instead of accessing the database. However, then, no guarantees could be given in regard to the freshness or the consistency of the data.

Therefore, we envision an architecture, where the AS layer is responsible for replica control (Figure 2). There are several AS instances, and each is connected to its own DBS which can be off-the-shelve software without replication semantics. Each application can run on several AS instances. The objects associated with an application are replicated across the AS instances. The AS layer performs replica control.

In principle, there does not seem to be a difference in architecture if the connection between the AS instances is a LAN or a WAN. However, the fundamental latency difference between LANs and WANs and the different purposes for which cluster replication and WAN replication are applied, require very different replica control solutions and we will discuss them shortly. In practice, large SOA are likely to integrate both cluster and WAN replication. At each location, a cluster solution distributes the load locally submitted. The different locations are then connected via a WAN replication solution.

3 REPLICATION STRATEGIES

Replica control has been well explored in the database community and a wide range of replica control solutions exist that have fundamental influence on non-functional properties such as availability, data consistency, reliability as well as system performance. Although the replica control algorithms themselves are not directly applicable to AS replication, the design space is similar at both layers.

3.1 Categorization

Wiesman et al. (Wiesmann et al., 2000) categorizes replica control algorithms by four parameters, partially taken from (Gray et al., 1996), developing a replication hierarchy.

Architecture. In a *primary copy approach*, each data item has a primary copy. Updates are always first executed on the primary copy which is responsible of propagating any changes to the secondary copies. Different data items might have their primary copies at different sites. However, often a single DBS replica has the primary copy of all data items. This makes concurrency control easier: this primary site determines the serialization order and all others apply changes according to this order. However, the site can become a bottleneck. In an *update everywhere* approach, each site accepts update transactions and propagates updates to other sites. This makes the approach more flexible since clients can submit their requests anywhere. However concurrency control requires now complex coordination among the replicas. In the context of AS replication, an additional challenge is that the different AS instances might cache different data (the most recently used) while the remaining data only resides in the database. This makes coordination more difficult.

Synchronization. *Eager replication* synchronizes all replicas within the context of the transactions. There are no stale replicas. However, communication and traffic complexity is high. In contrast, in *lazy replication*, a transaction with all its updates is first executed at one site, and then the changes are propagated to other replicas in a separate transaction. Transaction response time is shorter but the data copies are not always consistent. While eager replication is, in principle, more desirable than lazy replication, the potential response time increase might not be acceptable in WAN environments. Also, eager replication, if not designed carefully, might more easily lead to unavailabilities. However, eager replication is likely to be feasible in cluster replication.

Interaction. The third parameter *interaction* decides how often synchronization is done. In a *linear interaction* approach, typically each write operation within a transaction leads to a communication step. In contrast, in *constant interaction* the number of messages exchanged does not depend on the number of operations in a transaction. While constant interaction is essential in WAN environments where communication is likely to be the main bottleneck, it is less crucial in LAN environments.

Termination. The final parameter decides on whether a transaction is terminated via *voting* or *non-voting*. Voting requires an additional exchange of messages in order to guarantee that all replicas either commit or abort the transaction.

AS replication solutions can be categorized by similar means, and we can use the categorization to be aware of the solution spectrum that is possible. However, two things have to be considered. First, one has

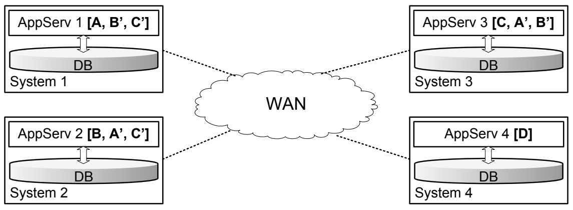


Figure 3: Primary Copy Replication on Object Level.

to be aware that the protocols have to be adjusted to work at the middle-tier layer. Secondly, we cannot simply assume that the general performance of the different algorithms will be the same as if they were implemented at the database level, given the particularities of AS replication.

3.2 Example Algorithm

We now describe one AS replication protocol in detail. We later use this algorithm to explain how an algorithm can be integrated into the simulator framework. Our algorithm is designed for a WAN environment where communication latencies are high.

3.2.1 Lazy Primary Copy

The algorithm we describe is lazy. This seems the most attractive in a WAN to avoid long client response times. We choose a primary copy approach since it simplifies concurrency control. Concurrency control at the AS layer is not yet well explored, doing it in a distributed and replicated setting would be even more challenging. Furthermore, the inconsistencies that can occur in a lazy environment if two different AS instances update the same item concurrently can quickly become a nightmare. A lazy approach makes constant interaction a natural choice because all updates are known at commit time, and thus, before propagation is initiated. Furthermore, voting can be avoided if the serialization order for each transaction can be determined by a single site. This is possible if all data items updated by this transaction have their primary copies at the same site.

Lazy replication provides only then fast response times for updates if the submitting client is close to the primary site. In order to keep the number of remote accesses low, the primary copy of a data item should be located at a site that is close to most of the clients that want to update this data item. For instance, if the data item refers to a specific client, then the primary copy should reside on the AS that is close to the client's home location.

3.2.2 Replication on Object Level

The natural granularity for replication in an AS environment is a business object (BO) because it builds a logical unit within the business semantics. Note that this might be quite different to the logical unit in the database, i.e., a database record. Therefore we refer to our approach as *primary copy on the object level*. Figure 3 shows an example setup. There are four AS instances with their local DBS connected via a WAN. There are four business objects with primary copies A, B, C, D, and the respective secondary copies A', B', C' (D does not have a secondary copy). Each of the four AS instances is the primary for one of the BOs. The assignment of BO can be based on various conditions e.g., semantical or geographical. We assume that each transaction only updates BOs for which a single site has the primary copies.

3.2.3 Protocol Description

We assume that each client request triggers a transaction within the AS. The transaction can consist of several read and write operations accessing local BO copies. The protocol proceeds as follows. When a client submits a read-only transaction to an AS instance, the local instance executes the request locally and returns to the client. No communication with other AS instances is needed. If the client submits an update transaction, e.g., updating A in the figure, if the AS instance has the primary copies of the *BOs* updated by the transaction (i.e., if it is system 1), the transaction executes and commits locally. The AS instance returns the result to the client and then forwards the update on the BOs to the secondary copies. If an update transaction is forwarded to a secondary copy (e.g., system 3), the secondary could either reject the transaction or transparently redirect it to the primary. The primary executes it and then forwards the changes and the response to the secondary (who sends them to the client) and all other secondaries.

The details of what exactly is forwarded (e.g., entire object, only changes, or the operation which then needs to be reexecuted at the secondaries), and how this forwarding takes place (via special communication mechanism or via web-services itself), might have a big impact on the performance. The simulation framework that we present in the next section is flexible enough to model such differences.

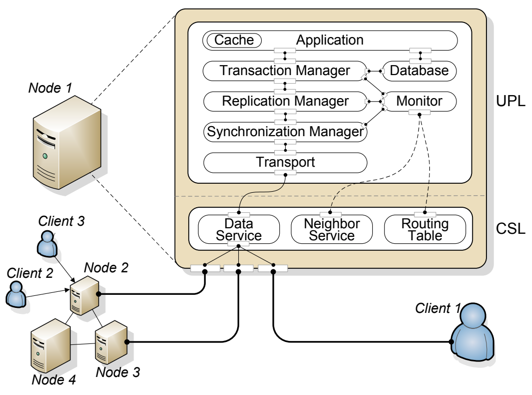


Figure 4: Node Architecture.

4 SIMULATION

A simulation, compared to a real implementation, has several advantages. Firstly, implementation is fast and does not require a real distributed environment as testbed. In contrast, implementing even a prototype replication tool into a real SOA is very complex, making it hard to implement several algorithms within a reasonable time frame. Secondly, prototype implementations are often not optimized, leading to bottlenecks and overheads that would not occur in any well-engineered implementation. A simulation can abstract from such artifacts. Finally, a simulation framework allows for a greater variety of parameters, and thus, a more versatile evaluation comparison. Therefore, we have designed a simulation architecture that captures the most important aspects of SOA, in particular in regard to performance implications. We show along one example how an algorithm can be implemented into our simulation framework, and how performance results can be derived showing the influence of various parameters.

4.1 Simulation Architecture

Our framework is based on J-Sim (Sobeih et al., 2006), a component-based simulation environment. J-Sim offers the concept of a *core service layer* (CSL) which implements already the basic services of a network layer. On top of this the *upper protocol layer* (UPL) provides the infrastructure for the replica control algorithms and emulates the standard AS components. Our solution maintains the component-based, loosely coupled architecture of J-Sim, and thus, is easily customizable and extensible. Figure 4 illustrates the architecture of one individual node (AS instance plus database).

4.1.1 Components of UPL

The `Transaction Manager` component is responsible for transaction management. The `Application` component simulates the business logic and maintains the BOs. It has its own `Cache`. BOs can be created, read, updated and deleted. The AS is also connected to the `Database` for the persistence of BOs. These three components are also found in a non-replicated multi-tier architecture. The other components implement the replication functionality. They basically intercept requests from clients and responses sent to clients in order to control the replication process. The `Transport` component, implements the transport protocol, e.g., TCP. It is connected to the `Data Service` provided by the CSL of J-Sim. The next higher layer is the `Synchronization Manager` which is responsible to handle the propagation of replication messages. The `Replication Manager` component implements the main tasks and execution flow of the replica control algorithm, e.g., the lazy, primary copy scheme.The `Monitor` watches all components for monitoring reasons. It also contains concepts that are needed by all other replication components, e.g., routing tables. The architecture resembles the interceptor-based approach used to plug-in functionality into web- and application servers.

4.1.2 Resource Consumption

In order to measure and estimate the CPU usage, every component is additionally equipped with a time module. Each time the component is triggered to do something, the time module collects the time of processing and relatively computes a percentage that mirrors the CPU usage. That is, we do not simulate time but measure the actual time the execution requires.

4.1.3 Topology

The above node architecture is used by all servers (both primary and secondaries in the example protocol). The nodes are connected through the port connected to the `Data Service`. Clients use the same port to send their requests. (e.g., `Client 1` to `Node 1`). A client is always connected to one of the servers.

4.2 Execution

Clients generate requests and send them to the server to which they are connected. We show here the execution flow using the implemented lazy, primary copy approach on object level. Other protocols will follow a similar flow but perform different actions at different time-ponts. A request is either a read-only or update transaction accessing several BOs.

Let's first look at a read-only transaction. A client submits the request in form of a byte stream to its local server. The request goes through the data service and is forwarded to the transport manager which implements a TCP abstraction. Thus, it models sufficiently well a web service request over a standard network and transport layer. The transport layer transforms the byte stream back into a request. The request goes through the synchronization mechanism (which usually does not have anything to do for client requests). Then it goes to the replication manager (nothing has to be done for a read request). When the request arrives at the transaction manager, a new transaction is started and the operations are submitted to the application layer which simulates some execution. If the requested BOs are not in the cache, they have to be loaded from the database, otherwise they can directly be read from the cache. After completion, the response moves back through the different layers. The transaction manager commits the transaction, the other layers don't have to do anything for a read request, and the client receives the answer.

Let's now discuss the steps when the client submits an update transaction directly to the server that contains the primary copies. As the request moves up the layers, nothing has to be done by the synchronization and replication managers since the approach is lazy and replication tasks are triggered after commit (eager replication might already perform some actions at this time point). The transaction manager again starts a transaction and the application simulates the processing of the write request. At the latest at transaction commit time the changes to the BOs are written to the database. When the result is returned to the replication manager, it detects a change state. It receives via the monitor all secondaries of the updated BOs (this meta-information is maintained by the monitor). It initiates via the synchronization manager the propagation of the state changes to these secondaries. At the same time, the result is also returned to the client. Secondaries receive the propagation messages also through the data service. The synchronization manager at the secondary determines that this is a propagation message (and no client request). An appropriate refresh transaction is started at the secondary to apply the changes on the BO copies.

In case the client submits a write request to a server that does not have the primary copies, the replication manager catches this request and it is forwarded to the primary which executes it as above. When the secondary receives the refresh transaction from the primary, the synchronization manager sends the response to the client and at the same time forwards the update upwards so that it is applied on the BO copies and in the database.

Table 1: Overview of Parameters.

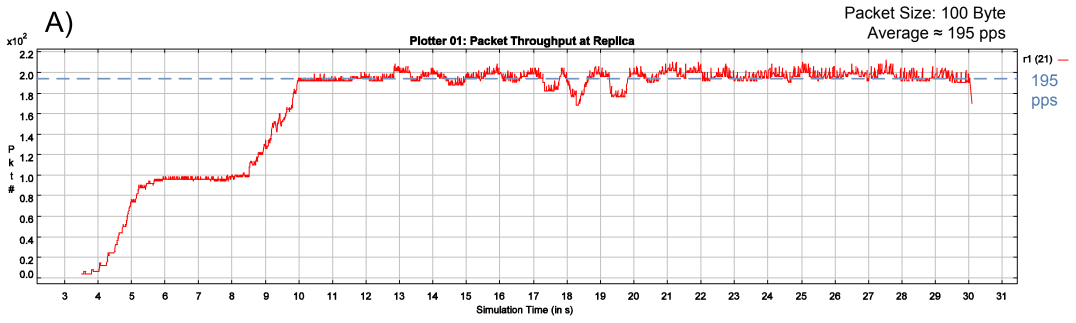
Domain	**Parameter**
System	N_S: Number of Servers N_C: Number of simultaneous Clients N_O: Total Number of Objects R_F: Replication Factor
Network	B_W/B_L: Bandwidth WAN/LAN L_W/L_L: Latency WAN/LAN
Transport	P: Packet Size MTU: Maximum Transmission Unit MSS: Maximum Segment Size
Transaction	C_I Interval of Client Transactions $T_{R/W}$: Read/Write Ratio N_O: Number of Operations $T_{/C}$: Transaction per Client
Database	D_R: Delay Read, D_W: Delay Write
Replication	R_A: Replication Algorithm

4.3 Parameters

The behavior of an algorithm is influenced by many parameters. Our simulation framework should help evaluating the system under varying conditions and understand the implications of each parameter. So far, we focus on the impact on performance based on the parameters depicted in Table 1.[1] N_S is the number of servers in the system. Each server can have a different number of clients, thus, N_C is actually a list of size N_S, indicating the number of clients for each server. The total number of BOs in the system is N_O while R_F depicts the replication factor, i.e., the number of copies per object. If $R_F = N_S$ than all objects are replicated at all servers (full replication), otherwise the copies are equally distributed across the servers.

We distinguish the bandwidth and latency of WAN (B_W, L_W) and LAN (B_L, B_W) messages. The system takes as input a topology graph that indicates whether links are WAN or LAN. For instance, all client/server links could be LAN while all server/server links are WAN. The transport layer has the typical TCP related parameters. For transactions, the overall load in the system is determined by the time C_I between two consecutive transactions submitted by an individual client. Furthermore, each transaction has a number of operations N_O and a ratio $T_{R/W}$ of read operations

[1]Semantics is also influenced by the parameters. For instance, using a lazy, update everywhere scheme in a WAN with high message is likely to lead to higher inconsistencies than in a LAN where updates are propagated faster.



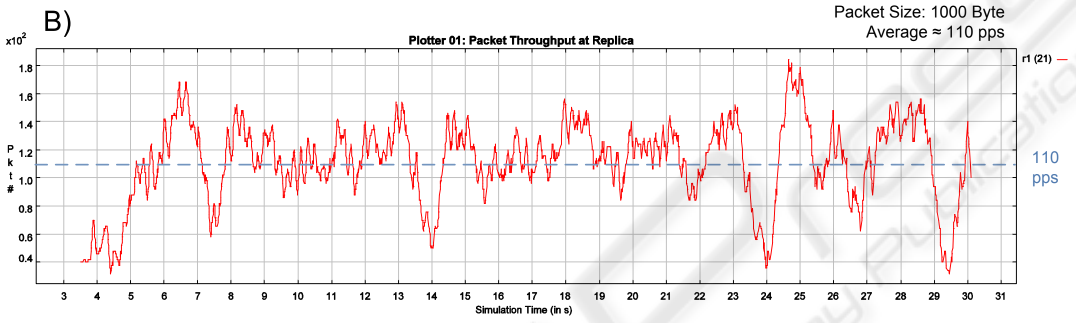


Figure 5: Packet Throughput.

among all its operations. Each operation accesses one of the BOs randomly. When executing an operation in the application, each read and write causes a delay for the database access (D_R and D_W).

If behavior over time should be measured, one can indicate $T_{/C}$ as the number of transactions per client until the simulation terminates. If averages should be calculated (e.g., average response time), the simulation runs until a confidence interval is achieved. Finally, the chosen replication algorithm is an important parameter influencing the performance.

5 SELECTED RESULTS

In this section, we shortly show how our simulator can be used to analyze the performance using the lazy, primary copy implementation. Figure 5 shows the impact of packet size assembled for transmission over the network. The experiment shows how a rather low level parameter can have a tremendous effect on performance. There is one primary and one secondary server. Each server has a single client submitting a transaction every two seconds. Each transaction has three write operations and two read operations. Each write operation triggers a change on a BO resulting in 5000 Bytes that have to be transferred to the secondary. We distinguish packet sizes of 100 Bytes (Workload A) and packet sizes of 1000 Bytes (Workload B). This means, the application message (15000 Bytes) has to be split into several network messages.

The figure shows the packet throughput for 100 and 1000 Bytes packet sizes over a time period of 30 seconds. A packet size of 100 Bytes leads to a rather steady amount of 195 *pps* (packets per second) with a variance of roughly $\pm 10\,pps$. In contrast, 1000 Bytes leads to only $110\,pps$ with a much higher variance. Having a small packet size where the application content is split in many network messages leads to a steady stream of small messages. In contrast, large package sizes lead to a much more bursty behavior since message exchange is concentrated to the time periods the application wants to send a message. Also, considerably less messages are sent in total.

Summarized, a small packet size produces an even, continuous and high packet stream which keeps network and server busy. However, peaks are not as high as with large message sizes. If the system is not able to manage the peaks created by large message sizes, small message sizes are preferable. The Byte throughput (bytes sent per time unit) behave similarly.

6 RELATED WORK

In the last decade, many different database replication strategies have been proposed (Pacitti et al., 1999; Pedone et al., 2003; Lin et al., 2005; Plattner et al., 2008). Some of them integrate replica control directly into the database kernel while others use a middleware-based approach.

In J2EE components like session and entity beans keep the application state. The invocation of objects is intercepted by the server and transactions are handled by the transaction manager. Basically all application events are exposed to the application server, making a non-intrusive integration of an replication algorithm feasible. ADAPT (Babaoglu et al., 2004) is a framework for application server replication providing an abstract view of the J2EE components in the server. It is based on the JBoss AS. Nevertheless, implementing a single algorithm using the framework is still likely to take months of developments.

Most industrial (such as WebSphere, Weblogic or JBoss) and many research solutions for application server replication (Felber and Narasimhan, 2002; Narasimhan et al., 2001; Barga et al., 2002; Wu and Kemme, 2005) use the primary copy approach. In many cases, however, the AS replicas share a single database and/or the business objects are not replicated. The database cache is often inactivated. The approaches are mainly designed for fault-tolerance. (Salas et al., 2006; Perez-Sorrosal et al., 2007) provide replication of business objects and represent very specific algorithms.

7 CONCLUSIONS

This paper discusses replication alternatives for business objects at the application server layer. We present a simulation framework that allows for a fast initial comparison of replication solutions before integrating them into an industrial system. In our future work we are planning to perform a thorough comparison of algorithm alternatives and the impact of network configurations and workloads. We aim at developing an online advising system, making suggestions for reconfiguration, in order, e.g., to handle a change in workload.

REFERENCES

Babaoglu, Ö., Bartoli, A., Maverick, V., Patarin, S., Vuckovic, J., and Wu, H. (2004). A framework for prototyping J2EE replication algorithms. In *Int. Symp. on Distributed Objects and Applications (DOA)*.

Barga, R., Lomet, D., and Weikum, G. (2002). Recovery guarantees for general multi-tier applications. In *IEEE Int. Conf. on Data Engineering (ICDE)*.

Felber, P. and Narasimhan, P. (2002). Reconciling replication and transactions for the end-to-end reliability of CORBA applications. In *Int. Symp. on Distributed Objects and Applications (DOA)*.

Frølund, S. and Guerraoui, R. (2002). e-transactions: End-to-end reliability for three-tier architectures. *IEEE Trans. Software Eng.*, 28(4):378–395.

Gray, J., Helland, P., O'Neil, P., and Shasha, D. (1996). The dangers of replication and a solution. In *SIGMOD Conf.*

Killijian, M.-O. and Fabre, J. C. (2000). Implementing a reflective fault-tolerant CORBA system. In *Int. Symp. on Reliable Distributed Systems (SRDS)*.

Lin, Y., Kemme, B., Patiño-Martínez, M., and Jiménez-Peris, R. (2005). Middleware based data replication providing snapshot isolation. In *SIGMOD Conf.*

Narasimhan, P., Moser, L. E., and Melliar-Smith, P. M. (2001). State synchronization and recovery for strongly consistent replicated CORBA objects. In *Int. Conf. on Dependable Systems and Networks (DSN)*.

Othman, O., O'Ryan, C., and Schmidt, D. C. (2001). Strategies for CORBA middleware-based load balancing. In *IEEE(DS) Online*.

Pacitti, E., Minet, P., and Simon, E. (1999). Fast algorithm for maintaining replica consistency in lazy master replicated databases. In *Int. Conf. on Very Large Data Bases (VLDB)*.

Pedone, F., Guerraoui, R., and Schiper, A. (2003). The database state machine approach. *Distributed and Parallel Databases*, 14(1):71–98.

Perez-Sorrosal, F., Patiño-Martínez, M., Jiménez-Peris, R., and Kemme, B. (2007). Consistent and scalable cache replication for multi-tier J2EE applications. In *Int. Middleware Conf.*

Plattner, C., Alonso, G., and T.-Özsu, M. (2008). Extending DBMSs with satellite databases. *The VLDB Journal*.

Salas, J., Perez-Sorrosal, F., Patiño-Martínez, M., and Jiménez-Peris, R. (2006). WS-replication: a framework for highly available web services. *Int. WWW Conf.*

Sobeih, A., Hou, J. C., Kung, L.-C., Li, N., Zhang, H., Chen, W.-P., Tyan, H.-Y., , and Lim, H. (2006). J-sim: A simulation and emulation environment for wireless sensor networks. *IEEE Wireless Communications*, 13(4).

Wiesmann, M., Schiper, A., Pedone, F., Kemme, B., and Alonso, G. (2000). Database replication techniques: A three parameter classification. In *Int. Symp. on Reliable Distributed Systems (SRDS)*.

Wu, H. and Kemme, B. (2005). Fault-tolerance for stateful application servers in the presence of advanced transactions patterns. In *Int. Symp. on Reliable Distributed Systems (SRDS)*.