Computing and Informatics, Vol. 34, 2015, 973–995

EVALUATION OF DATABASE REPLICATION TECHNIQUES FOR CLOUD SYSTEMS

Melissa Santana, José Enrique Armendáriz-Iñigo

Depto. de Ingeniería Matemática e Informática
Universidad Pública de Navarra
31006 Pamplona, Spain
e-mail: santana.66548@e.unavarra.es, enrique.armendariz@unavarra.es

Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universitat Politècnica de València
46022 Valencia, Spain
e-mail: fmunyoz@iti.upv.es

Abstract. Cloud computing is becoming one of the preferred paradigms to deploy highly available and scalable systems. These systems usually demand the management of huge amounts of data, which cannot be solved with traditional database systems. Traditional replication protocols are not scalable enough for a cloud environment. This paper evolves different static replication techniques to achieve transactional support providing high availability and scalability as needed in cloud systems. This proposal offers different consistency levels according to the demands of client applications using a replication strategy based on a combination of traditional replication techniques with asynchronous epidemic updates. We have run several simulations that show this is an interesting approach to provide transactional support to clients with different consistency guaranties while leveraging the resources used.

Keywords: Database replication, performance evaluation, scalable systems, cloud systems, distributed systems

Mathematics Subject Classification 2010: 68M14, 68P20

1 INTRODUCTION

Replicated databases have become attractive due to an increasing demand for storage systems which provide high scalability and availability. However, replication presents the problem of keeping all data stored in all replicas consistent. There are several replication protocols to minimize this consistency problem which propose different ways to maintain the data updated in all replicas using some communication tools to propagate the changes. Thus, to reduce the cost of propagating updates to all replicas, two different strategies have been proposed according to its synchronization [1]: eager replication and lazy replication. Eager replication performs the synchronization within transaction boundaries; while lazy replication propagates the changes after the client receives the transaction commit. Eager protocols comply with the consistency requirement as part of their execution but introduce a higher latency. On the other hand, lazy protocols lead to inconsistencies, which have to be solved later.

With regard to where updates are carried out we distinguish between primary copy [2, 3] and update everywhere [4, 6]. In a primary copy variant all update transactions are executed in the same replica, the delegate. In this case the synchronization is performed in a straightforward way, propagating the delegate updates to the secondary replicas. For update everywhere the update transactions can be executed in any replica; this implementation presents some issues like conflicts between transactions belonging to different replicas.

Group communication systems have provided a useful abstraction to be employed when designing update everywhere replication protocols [7, 8, 9]. There are several replications techniques based on one of the primitives of group communication: total order broadcast [1]. The first technique is active replication, which executes the update in the delegate server that issues the total order broadcast on behalf of the client. The second technique is certification-based replication. It is optimistic; i.e, operations contained inside a transaction are executed with no restrictions and check for consistency violations at its end. In this technique all servers execute a certification phase where it is decided if the transaction can commit or must abort. And the third technique is weak voting replication, which is similar to certification-based replication. The main difference is that the certification phase is replaced by a weak voting phase, where the delegate takes the decision to commit or abort.

Because of the scalability limitations of traditional database replication, there is a new concept which brings highly scalable database at low cost. This is called NoSQL (Not only SQL) [10, 11] and is known as one of the best cloud-storage resources. In terms of scalability, the cloud offers concepts like (virtually) infinite scalability and services on demand [12].

There are some specific scenarios where the ACID (Atomicity, consistency, isolation and durability) properties can be relaxed, and high performance and availability levels can be achieved. Examples of these cases are the key-value data store provided by Amazon called Dynamo [10], the Bigtable from Google [14] or PNUTS

by Yahoo! [15]. However there are applications that cannot get the benefits of the cloud-storage due to their transactional nature. Considering an on-line sales web application, there are data that can be accessed outside the context of a transaction (e.g., checking the warehouse for the amount of units of a certain item in order to estimate if they have to order more units of that item to their supplier) while other data do not (e.g, a customer buying the previously mentioned item or the company buying items to the supplier). Another example is home rental businesses, the pictures of the rooms of a house are hardly changed nor there is a need to see its latest comments' feedback, as opposed to a client's deposit on a house. Hence, there is a need for serializability transactional behavior on deposits while this can be relaxed or skipped at all in the rest of operations.

The cloud offers a pay per use service [12], and for this reason it is important that cloud systems only use the amount of services that they require [13]. This document presents a performance comparison of the aforementioned replication techniques adapted to the cloud environment. We have tried to maintain data consistency while keeping high availability and scalability.

Current cloud systems are generally classified [16] as *Infrastructure-as-a-Service* (IaaS), *Platform-as-a-Service* (PaaS) and *Software-as-a-Service* (SaaS) systems. An IaaS system manages physical resources (computing cycles, storage, communication bandwidth, ...) and, through virtualization, this system is able to provide a logical infrastructure to their customers. The latter are responsible for deploying all the software stack (including the operating system and distributed middleware) in order to build their applications. A PaaS system provides a higher abstractions level. Instead of providing a virtual infrastructure, it provides an elastic programmable platform where applications can be easily built. Finally, in a SaaS system a given application is being provided to the customers, who are also the direct users of those applications.

The aim of this work is to adapt the classical database replication techniques to a cloud environment. To this end, our objective is to follow a *Database-as-a-Service* (DaaS) paradigm, as a specialized case of a PaaS system; i.e., the interface and service being provided by a system of this kind is that of a (scalable and adaptive) relational database and this partially matches the aim of a PaaS system.

Unfortunately, it is not possible to apply those classical techniques directly, so it is necessary to partition the data previously and to distribute the resulting partitions in a subset of M replicas (having $M << N$, with N the total number of replicas). We present a two level replication behavior. The first level consists in a small number of P replicas (with $P << M$) which will have the strongest consistency management [2] and will maintain the more recent versions of the data. Each one of the P replicas will propagate the changes asynchronously to the next level. We can iterate this process to several levels and, thus, we did implement a replication hierarchy tree with the most recent versions at top whereas older versions are at the leaves.

We provide different freshness level to each transaction, i.e. each client decides the freshness level that they need for each transaction. In this way, if a transaction has the highest freshness level it will be executed in the first level of the hierarchy.

On the contrary, if the freshness is lower, then an older data from the hierarchy can be returned. The hierarchy level will depend again on the freshness constraints and the amount of resources.

Finally, we execute transactions with a rational use of resources. Cloud systems are based on a pay-per-use principle. So, we have designed a mechanism that shutdowns database replicas that are not being used, restarting them only when necessary.

The rest of this document is structured as follows. Section 2 details the replication technique being implemented. Section 3 presents the different experiments and their results. Finally, Section 4 shows the conclusions.

2 REPLICATED DATABASE SYSTEM DESIGN

This section describes the system proposed in this work. It simulates the replication techniques previously explained combined with a cloud scheme to provide high scalability, availability and an appropriate use of resources. Section 2.1 explains the motivation. Section 2.2 provides a system overview. Later, Section 2.3 describes the system model.

2.1 Motivation

It is well known that the performance of replication protocols depends on the workload characteristics. For instance, the primary copy can provide a higher throughput in a replicated system where most transactions are read-only [2]. On the other hand, if there are many update transactions, an update everywhere approach based on total order broadcast may be more appropriate [1].

Techniques based on update everywhere present low scalability because of the costs of propagating the changes made by transactions in total order broadcast and applying them in remote replicas. This scalability disadvantage can be alleviated by creating a hierarchical server structure for update propagation. In this case, an update everywhere protocol would be combined with a primary copy protocol. Let us say that there are S servers (one database partition per server), and each one has N children. These S servers receive the transactions from the clients, each server behaves according to the update everywhere protocol implemented and the response is sent to the client. Then, the transactions are sent by these servers asynchronously to each child, having a primary copy behavior. And this can iteratively go on so that it could exist as many hierarchical levels as needed. This causes that at a particular moment the replicas in the lowest level of the hierarchy have an older version of the data and the lower is the level in the hierarchy the older is the version.

When a transaction is received it can be forwarded to a secondary replica depending on the freshness level required, achieving a better load balancing and scalability when compared to previous solutions. A further advantage of the presented architecture is that system replicas can be upgraded or downgraded in the hierarchy according to current system requirements.

2.2 System Overview

Bearing in mind the previous ideas, we have developed a simulator that implements the replication techniques explained in this work. Recall that these replication techniques were evaluated in [1] and the results obtained assumed a static system. For this work we propose to evaluate their performance for a cloud system having as the dynamic part to promote or degrade the replicas to achieve the load balance, as well as to allow executing the read-only transactions in a replica located in a lower level in the hierarchy depending on the data freshness required. Furthermore, we propose to turn off or turn on the replicas to consume the lowest amount of cloud resources.

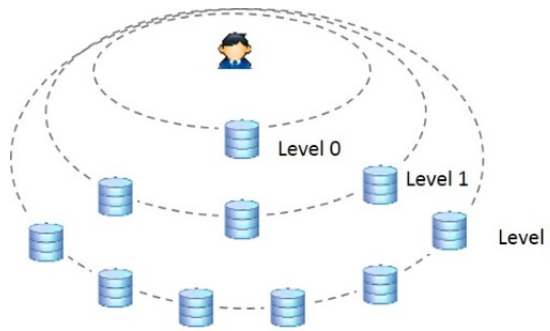


Figure 1. Proposed structure for the servers

The system consists in a partially replicated database. To this end, it is necessary to have a component which controls the database partitions [18, 19] and creates the hierarchy mentioned that results in the structure shown in Figure 1. The servers in Level 0 take the behavior specified by the replication technique implemented (one of those mentioned in the introduction). To the rest of levels the protocol we apply is a lazy primary copy to transfer the changes in an asynchronous way. In case that a client sends an update transaction, the transaction is executed by a replica in Level 0 managed by the replication technique considered. After all the process is completed and the client receives the response, the changes are propagated to the replicas in Level 1 (lazy primary copy protocol). In the same way, the replicas in the Level 1 will propagate the changes to the replicas in Level 2 asynchronously and so on until the replication depth for that partition is reached.

With this structure, it is possible to have different versions of the same data in one partition. Hence, the read-only transactions play the lead role regarding this hierarchy. Each one of these transactions have a freshness level where we assume that it can vary from 0 to 1 being the former the tightest one while the latter the most relaxed, respectively. Read-only transactions with strict freshness guarantees are forwarded to a replica in Level 0 while the rest are forwarded to other levels as we can see in Figure 2. On the left side of Figure 2 we have a transaction t with

a freshness of 1; this transaction has to be executed at the top of the hierarchy (Level 0). Whereas t with lower freshness requirements, as seen on the right side of Figure 2, can be forwarded to any of the replicas underneath.

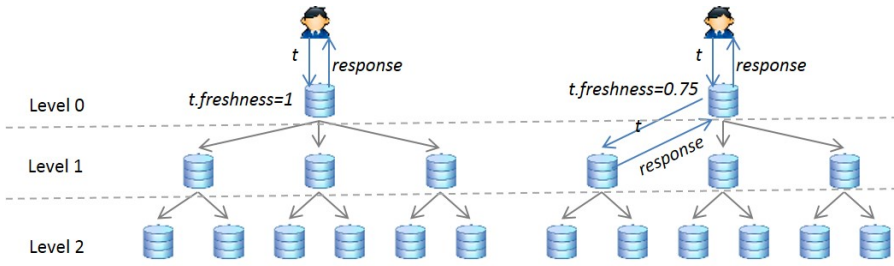


Figure 2. Transaction execution according with the freshness level

Ideally, we may think that this propagation process through the hierarchy could be extended to the infinity. However, we are using a pay-per-use model and have to make a judicious use of resources and locate them through the levels smartly enough to obtain the best performance. This is an optimization problem and falls out of the scope of this work. In this paper we present a service that switches on and off replicas from a scenario with a fixed number of replicas; initially, not all of them are up. To this end, we have a component that monitors the behavior of all replicas in the system. Hence, we have a Metadata Manager (MM) similar to the ElasTraS proposal [5]. Let us see this with an example of the resources' management with one data partition as shown in Figure 3. Initially, we only have active one replica at level 0 (see Figure 3 a)). When the metadata manager detects a relevant increase in the number of requests per second, it adds one replica from level 1 to start accepting read-only transactions that can alleviate the load supported at level 0 (Figures 3 b) and 3 c)). The workload can increase further and then the MM can notify more replicas to join in (Figures 3 d) and 3 e)). Finally, scenario V represents a situation in which the MM decides to remove one of the replicas upon detecting an important decrease in the number of requests in order to save resources (Figures 3 f)–3 i)).

2.2.1 System Model

Figure 4 shows an abstraction of our cloud replication proposal. We will simulate this system according to a discrete event simulator. We have developed it using version 2.34 of the NS2 network simulator which was created using C++ code and otcl as scripting language. The main components of the system are:

Client Applications. Clients are sources of transactions. A client submits the transaction and receives the response from the server. These events are repeated following certain parameters and are controlled by a timer. A client only submits one transaction at a time, waits for its outcome and, when scheduled, submits

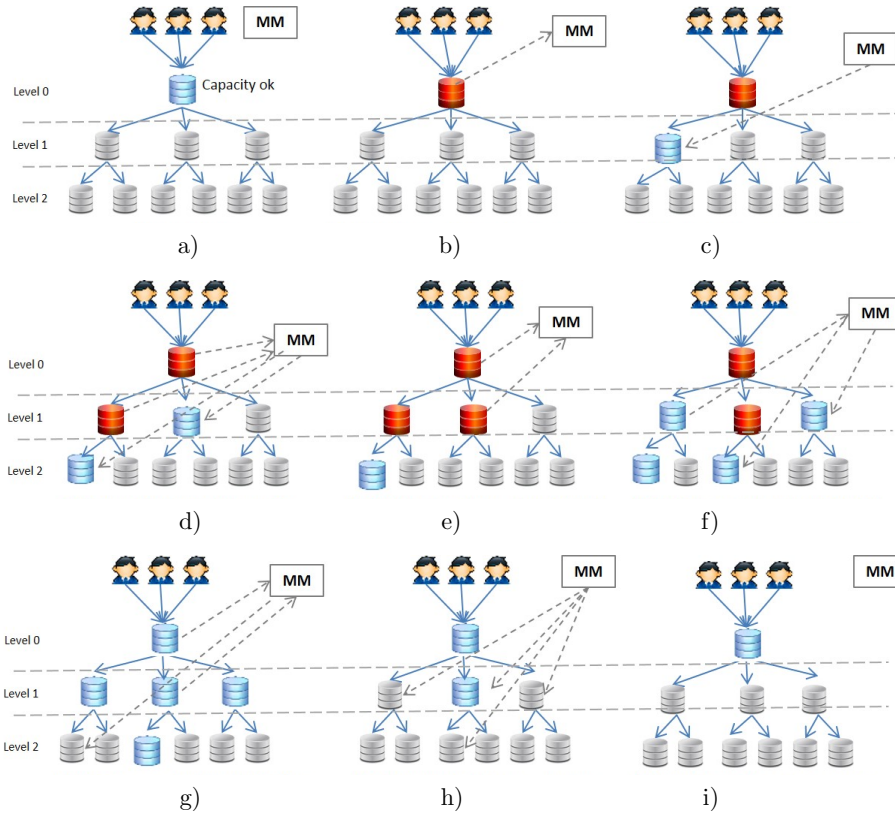


Figure 3. Resources use for the proposed system: a) Initial state, b) Maximum capacity, Send notification, c) Activate node from the next level, d) Still maximum capacity, activate another node in the next level, e) Maximum capacity, Send notification, f) Minimum capacity, Send notification, g) Minimum capacity, Send notification, Shutdown server in lowest level, h) Shutdown the unnecessary servers, i) Shutdown the unnecessary servers

the next one. One server is connected to a subset of the total clients; clients are evenly distributed among all servers. Clients gather all the performance data and compute statistics.

Servers. Each of them maintains one database partition. Each server hosts a local database manager using a replication technique. To accomplish this, the server application is structured in three levels:

1. Communication module. It represents the interactions among servers and each server with its associated clients. There is one instance per server. This module allows the server to multicast messages and manages the multicast

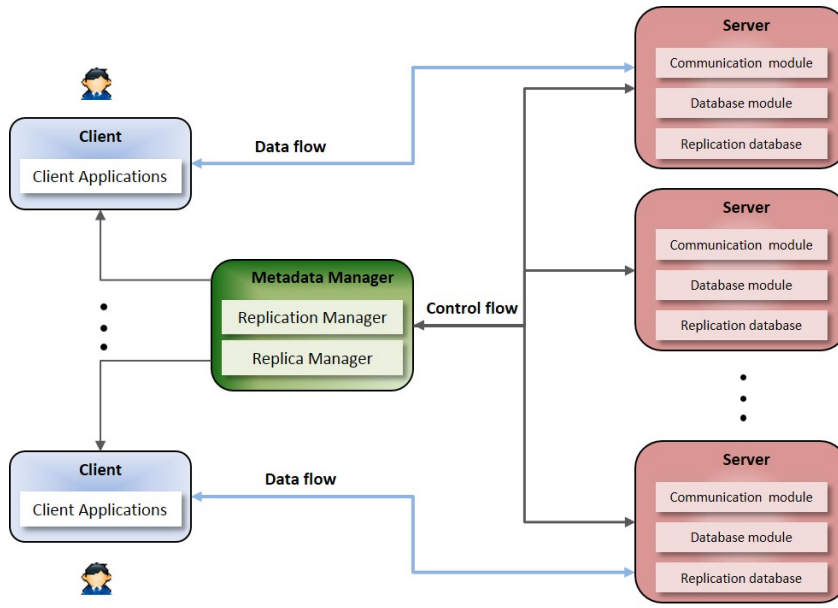


Figure 4. System model

group to which the server will communicate. It also controls the clients that are connected to the server. The message queues in each node are FIFO.

2. Database module. It simulates a database system. There is one for each server. This module includes a lock manager and a data manager. The lock manager supports all the concurrency needed to generate SI transactions in each server. The data manager handles operations to read and write data from/to the database. We assume that all the data is in main memory.
3. Database replication module. It runs the replication protocol instance on each server. The module behavior depends on the replication technique used in the simulation (primary copy, active, certification, weak-voting, etc.).

Metadata manager (MM). It stores the system state [5], viz., partition information, mapping of partitions to replicas, hierarchy for each partition. It deals with failures, and monitors the health of the system. When a server reaches its maximum capacity, it notifies the MM, so a child could be promoted. In the same way, when the server reaches the minimum load, the metadata will turn off one child at a time to save resources.

2.3 Potential Bottlenecks

At a glance, there are two components in this architecture that could become a performance bottleneck:

1. the MM, and
2. nodes at level 0 in the hierarchy.

Let us discuss how such a potential problem is being dealt with.

The MM component does not intervene in the clients' transactions as it does not reside in the data path. Besides, as partitioning is done to exploit data locality, clients can cache the metadata regarding the replicas where they have to connect as delegates so the MM-client interactions do not potentially generate a system bottleneck. On the other hand, the MM component has a passive role regarding monitoring. It only receives notifications from database replicas when their workload reaches some preconfigured upper and lower thresholds. Depending on the information received it decides whether a new replica needs to be added or an existing replica at a given layer should be passivated. Node passivation is not a long task. Once a request of this kind is received by a given replica, it stops receiving new updates and leaves its active role in the replication protocol. So, the MM only needs to send such a request and it is obeyed without requiring further intervention. Thus, the interaction between the MM and replicas does not constitute a bottleneck. Finally, as the MM is a virtually centralized component we need to establish a set of MM replicas where we keep the monitoring information up to date so that the MM is not a single point of failure. This needs to ensure an almost synchronous replication regarding data location and replication or replica monitoring. As the modification rate of this metadata is low, we can use Paxos as the replication protocol and with the previous issues covered we humbly think it does not become a system bottleneck (actually, Chubby uses this protocol).

We have to be specially cautious also about the addition of a new node in a given partition and hierarchy layer HL, such action requires a longer interval. The role of the MM is only to trigger the live migration between the donor node and the new node and choose both nodes; in other words, this process does not overcome a bottleneck at the MM. The live migration between nodes is described in [5] and consists in iteratively transferring the database cache and state of active transactions. Usually, the information to be transferred is the database cache. The new node to be added at level HL should be one from $\mathrm{HL}+1$ or, if not, from layer $\mathrm{HL}+2$ and promoted to layer $\mathrm{HL}+1$; the worst case will be the addition of a new node at the bottom layer. On the other hand, at the top layer, we need to add a node to level 0. In such a case, it is needed to be transferred the readsets and writesets of active transactions along with the database cache; otherwise, it is only needed to transfer the readset of active transactions (it is expected that readset will be made of scanning interval rows and only the start and end identifiers are needed). The end of the live migration process is known as atomic handover. If the

replica is at the core layer, the ownership of the active transactions is transferred to the new node. Moreover, the last unsynchronized state of the databases should be copied along with the state of active transactions. All these state transfers need to be completed without disruption (i.e., their effects are atomic).

The second potential bottleneck is the management of nodes in level 0 (i.e., the top-most level of the hierarchy). What happens when some nodes in the top-most level of the system hierarchy are overloaded? Recall that level-0 nodes are the managers of different database partitions. However, this does not necessarily imply that there is a single manager per partition (a primary copy approach). Indeed, there may be multiple level-0 nodes managing the same partition and balancing such partition workload among them (i.e., update everywhere). The solution consists in:

1. if the workload is not well-balanced among all level-0 partitions, applying a work-balancing approach; or,
2. if the workload is almost equal in all 0-level nodes, adding another node or changing the replication protocol.

Let us explain the latter in depth, if the protocol running on level 0 is a primary copy and the update ratio is increasing, the system must change to an update everywhere protocol so the update workload is evenly distributed among all nodes. Besides, if the protocol running was also an update everywhere replication protocol, we can add a new replica from the bottom layer, as we have previously described.

However, if this process continues then we can face that the system does not scale as this was the traditional solution so the system has to repartition the data. The MM will decide according to data access pattern and its location which partitions to merge. The pair of heaviest and lightest loaded partitions merge their data and later divide them again to be equally loaded. This process can be seen and getting replicas from lower layers and start the live migration for the new repartition. In this case, there is a lease of the ownership from the old core to the new core. From this moment on, the lower layers start receiving new updates from the new core of items they did not own before. Thus, they will have a Δ step of versions in these lower layers. In order to avoid a thrashing behavior when layer 0 is being reconfigured (i.e., repartitioned) the thresholds used for tagging a given set of partition-managers as overloaded or underloaded are set at very distant values (e.g., 80 % and 20 % of its full service capacity, as shown in Table 1).

Since repartitioning is only initiated in case of overloading, the initial system deployment should set a large number of partitions on the database being served. Our recommendation is to apply such partitioning action implying that the typical workload was close to 50 % of the maximum service capacity of each level-0 node.

3 EVALUATION

This section presents the performance evaluation of the proposed system. An extensive set of simulations have been run to compare which combination of all the

replication techniques along with the hierarchy performs the best. These simulations measure the transactions response time seen by clients as well as the transaction abort rate and system throughput. We repeated ten times each measurement with 10 000 transactions and each time the first and the last 10 % of transactions were discarded in each experiment. The database settings were based on numbers in the literature [1] and represent mean values. The simulation parameters, together with their meaning, are shown in Table 1.

The *total size* parameter is divided by the *copy size* parameter to obtain the number of partitions. In this case we had 25 partitions. Each partition has four associated replicas (*replica number* parameter); i.e., we have 100 servers in the simulation. The database has a hotspot whose size in rows is fixed by the *hotspot* parameter. Each transaction accesses to a given number of rows of the hotspot that is defined in the *hotspot rows* parameter. Every delegate server has 6 clients which gave us a total of 600 clients. The transactions size varies randomly between 10 and 20 operations. The response time of a transaction depends on the *write time* and *read time* parameters for each operation and the network latency (total order or FIFO multicast).

A combination of two different scenarios is considered: percentage of update transactions and the freshness degree of read-only transactions (i.e., *freshness*). With regard to the rate of update transactions (*update ratio* in Table 1), we have considered from a pure read-only scenario (0 % value in Table 1) to a write intensive scenario (75 %). On the other hand, read-only transactions could be configured so that they can jump through the hierarchy replication tree of each partition; a range from no transaction accepting old values (0 %) to transactions reading any available version of the data (100 %) has been considered.

A replication hierarchy depth up to three levels has been evaluated, where servers from the top of the hierarchy lazily send updates to the replication hierarchy every 0.4 ms (*update interval* in Table 1). Initially, both the non-hierarchical and the hierarchical protocols start with 4 replicas per partition. This implies that level-0 (the top-most layer) in the hierarchical management is served by 4 replicas. Each of the remaining layers only has a single initial replica. So, the hierarchical variant is initially deployed onto 6 nodes. The number of replicas in those other layers is varied dynamically, depending on the workload in each layer. The thresholds being considered are shown in Table 1: a new replica is created when the workload exceeds 80 % of the node full capacity and an existing replica is removed if the workload does not reach 20 % of the full serving capacity.

Although there are several standard benchmarks for database performance evaluation (TPC-C, TPC-E, YCSB), their dataset partitioning are not considered. Besides, we want to emphasize the advantage of our proposal in the presence of update intensive or conflict intensive scenarios when compared with traditional database replication techniques. As a result, a specific database scheme has been selected in our evaluation; actually, we have been inspired by the experimental setups provided in [1, 6]. To this end, vertical partitioning has been applied in the assumed database. Each partition consists of three relations, each one with two integer columns (one

of them is the primary key) and two varchar columns. The average record size is 200 bytes. Each table is filled with a large amount of records up to the configured partition size. Each write operation accesses a given range of sequential records from one of the partition tables. In order to raise some degree of conflicts among transactions, as commented above, a hotspot has been defined in a similar manner as presented in [6].

In order to validate the adequacy of the simulation parameters and its results, an initial configuration based on a single partition was tested and compared with the same deployment onto our MADIS [17] middleware (with 4 replicas and supporting the same workload, being tested with 100 TPS). Once the results were checked and matched, the simulation was extended with multiple partitions and a larger set of workloads whose results are presented in the sequel. Right now, we are developing a prototype with some preliminary results [20].

Parameter	Meaning	Value
Total size	Database size (GB)	750
Copy size	Size of each database partition (GB)	30
Number of replicas	Number of replicas for each partition	4
Protocol	Replication technique used for the simulation (primary copy, certification, etc.)	1–6
Client number	Number of clients in the simulation evenly distributed among servers	600
Update ratio	Rate of write operations in a transaction (%).	0, 25, 50, 75
Write time	Time for a write operation (μs)	20–120 μs
Read time	Time for a read operation (μs)	20–120 μs
Hotspot rows	Number of operations per transaction that go to the hotspot	3
Simulation time	Total time for the simulation in seconds	500–1 000
Network delay	Latency of the network	0.2 ms
Operations	Number of operations in one transaction	20
Hotspot	Number of records in the hotspot	2 000
Freshness	Willingness of accepting older versions (%)	25, 50, 75, 100
Update interval	Time between server actualization to its children	0.4 ms
Maximum capacity	Maximum capacity of server processing	80 %
Minimum capacity	Minimum capacity of server processing	20 %
Level number	Levels in the hierarchy	3

Table 1. System configuration

3.1 Results

Results are presented in the following order: response time, abortion rate and system throughput. For each of these measurements we are going to consider two

different scenarios with and without hierarchy. The latter represents the worst case scenario where all the workload, especially read-only transactions, have to be shared among all replicas, whereas the former represents the best scenario where read-only transactions can be split across the replication hierarchy (if suitable because of the *freshness* parameter).

The first metric (response time) tells us how fast the system responds to the client. Actually, we can consider that the best system would consist in a dumb protocol that aborts everything coming from clients. That is why it is so important to run the second metric, how many of the total submitted transaction ended up aborted by the protocol; i.e., we assume all transactions want to commit. In general, an optimal system would have the shortest response time with the lowest abortion rate. Finally, the throughput tells us how the system can handle the incoming workload. Ideally, if we injected a load of X Transactions Per Second (TPS) in the system, we expect that the system will handle this X TPS; nevertheless, it is often that due to concurrency issues and the replication protocol management the system cannot handle that rate. Thus, we look for a system with the best throughput.

To evaluate the system behavior in regard to the previous three aspects, we varied the rate of update transactions; with this, we can estimate the effect of propagating the updates to all replicas that is combined (or not) with the hierarchy. To achieve an increase in performance applying the hierarchical architecture, we added a freshness factor that tells us the percentage of transactions that can be executed in different replication levels.

If we vary the writing factor we have three scenarios: the first one is update intensive; the second is half of read-only transactions and the other half update transaction, and the third scenario consists in a read intensive load. This variation applies to both architectures with hierarchy and without it. In the case of having hierarchy, we varied another parameter: the freshness. This variation gave us the opportunity to make a correct evaluation of the performance gain applying the hierarchy.

3.1.1 Response Time Results

Results without hierarchy. Let us start with our results with no hierarchy; recall that these results represent the worst case scenario. Figure 5 shows the results for all the replication techniques explained above executed with different write factors. In general, with all replication techniques, we can infer that the greater the write factor is the higher the response time is. All the results are in line with those obtained in [2]. As expected, the active replication technique had the worst performance for the write factors applied. This is because the delegate server has to wait for the acknowledgment from all the replicas to send the response to the client; besides, all operations (including reads) are also sent to all replicas, so they have to execute as well taking more time. Following the trend of poor performance, a primary copy protocol shows a weaker performance in update

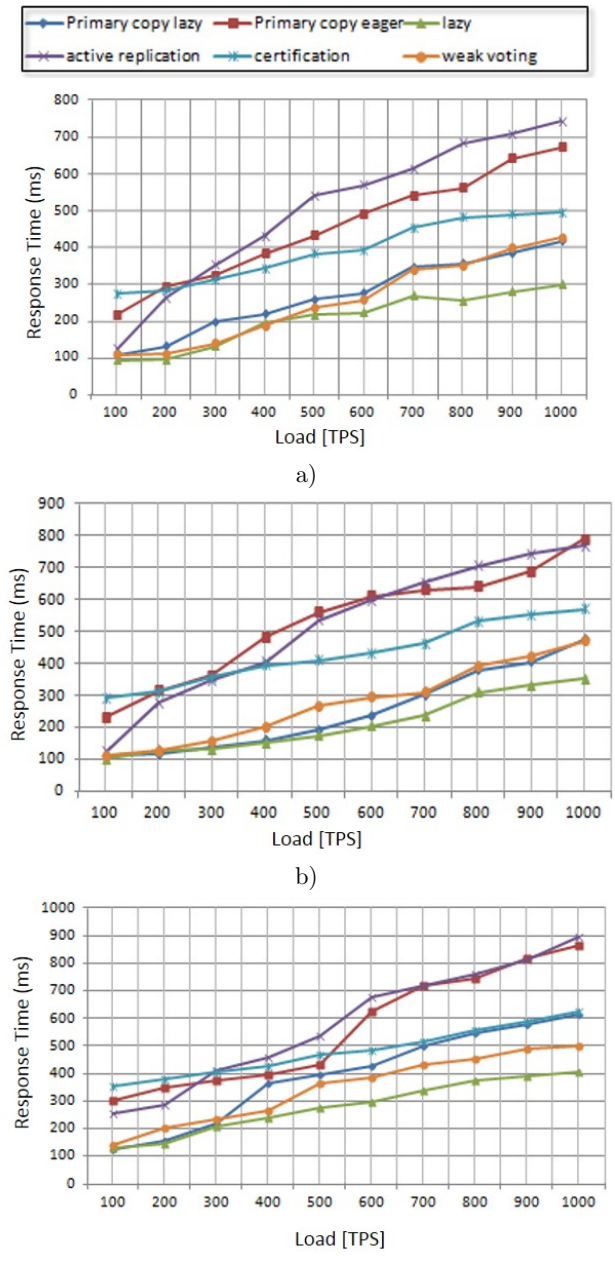


Figure 5. Non-hierarchy response time results; a) Write factor: 0.25 (without hierarchy) b) Write factor: 0.5 (without hierarchy) c) Write factor: 0.75 (without hierarchy)

intensive scenarios; later on, we will see the price we pay for that. On the other hand, the lazy protocol presents the best results here; this is caused by the fact that the delegate server does not have to wait for the replicas to answer. With a certification technique, the deterministic certification phase along with the total order broadcast allows the servers to assure that every transaction will have the same result in all replicas. However, its response times are higher than in a weak voting technique due to the processing required to certify that there is no conflict between another transaction and can commit, or in effect there is a conflict and must abort.

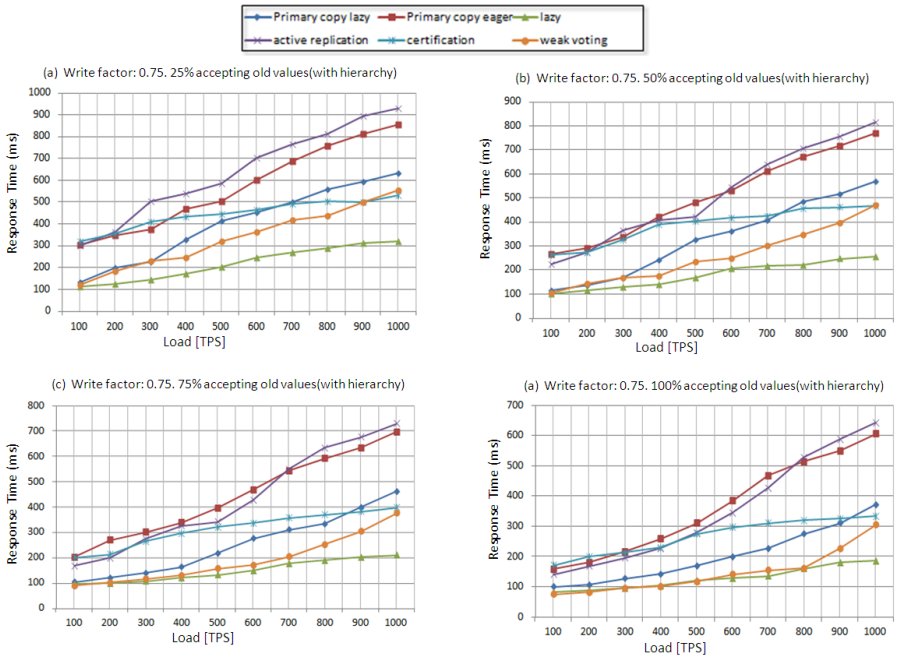


Figure 6. Response time with hierarchy depending on write factor (0.75) and freshness

Results with hierarchy. With the same configuration parameters that were used for the non-hierarchical simulations, we have run several simulations with the hierarchy added for a write factor of 75 % (update intensive) and different freshness parameters whose results are shown in Figure 6. As expected if we require strict versions of data then we will obtain similar results that those with no hierarchy at all (Figure 6 a) vs. Figure 5); this is because the freshness is not high enough to balance the high number of update transactions with the readings executed in lower levels. On the contrary, as seen in Figure 6 d), when we have no restrictions on versions we achieve a significant reduction in the response time. Finally, if we take a look at each replication technique on this latter figure

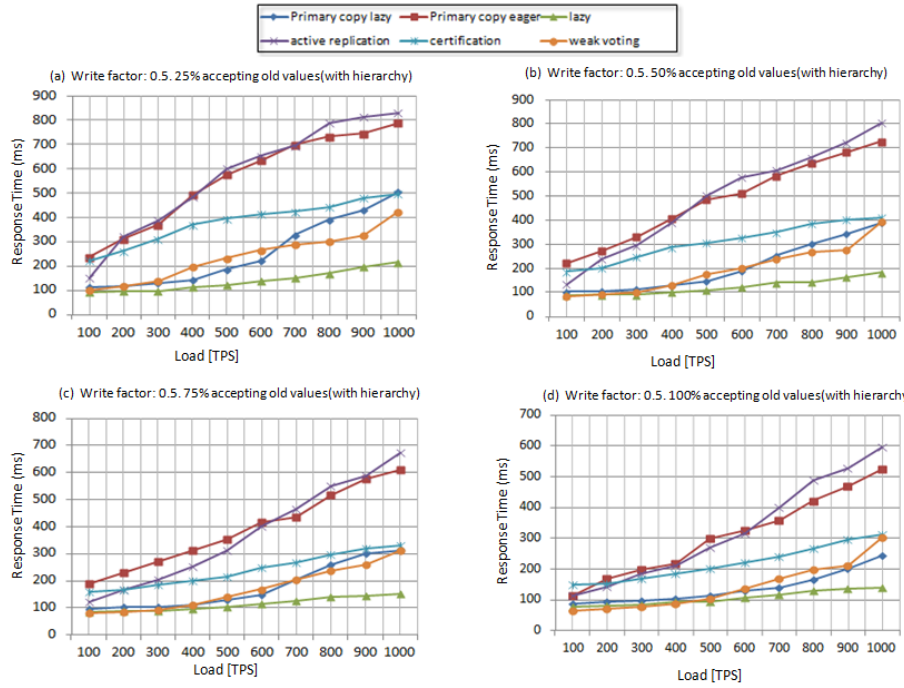


Figure 7. Response time with hierarchy depending on write factor (0.5) and freshness

with Figure 5 c) we realize that the behavior of all of them is similar in terms of their trends though their response time is reduced by approximately 30 %.

As expected, when the write factor was decreased (up to 50 %), the response times were decreased as well, as shown in Figure 7. On the other hand, Figure 8 shows the results for applying a write factor of 25 %. The performance obtained is the best of all implementations. This is important because for a cloud system the majority of operations are read-only transactions. Even in the worst case scenario of read-only transactions, i.e., only 25 % of them accept old values (Figure 8 a)) we obtain a significant gain in the performance against the same scenario with no hierarchy (see Figure 5 a)) or all the previous cases with hierarchy.

We can observe that in scenario with a low amount of update transactions (0.25) the different replication techniques have a similar behavior; this was expected because the transactions that take the major server processing time are those with write operations. To explain this we can observe Figure 8 d) which shows the response time for 100 % of read-only transactions accepting old values, certification technique and weak voting are very similar. The certification technique does not spend so much time in the certification phase and applying the remote updates since almost all read-only transactions execute in the lower levels of the hierarchy.

Finally, from all that we have said, it follows that Figure 8 d) represents the best case scenario and Figure 6 a) the worst case scenario of our experimental setup, respectively.

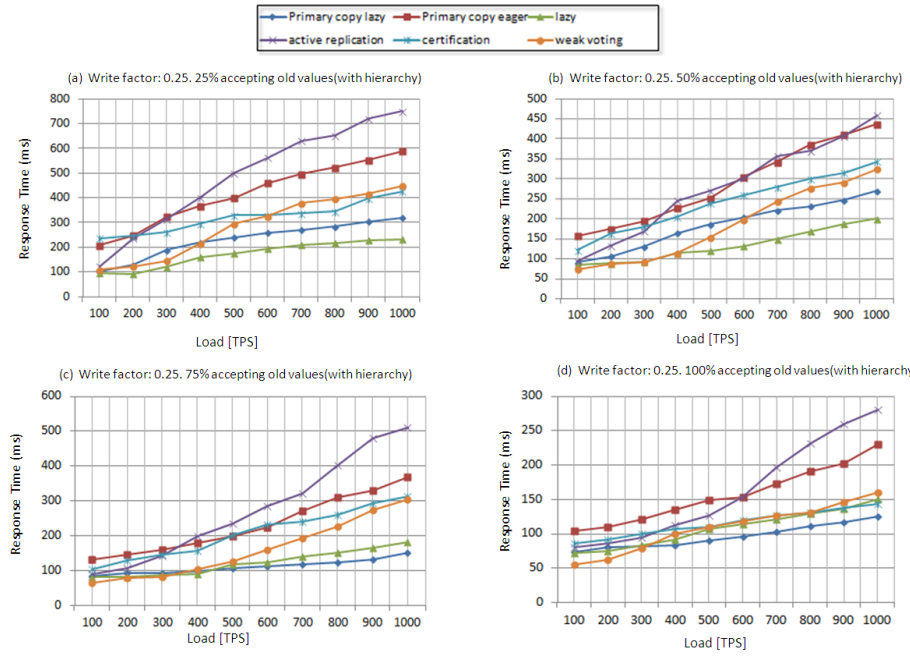
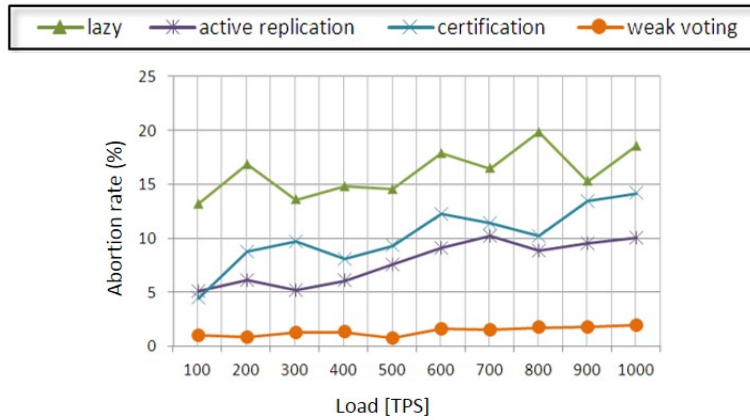


Figure 8. Response time with hierarchy depending on write factor (0.25) and freshness

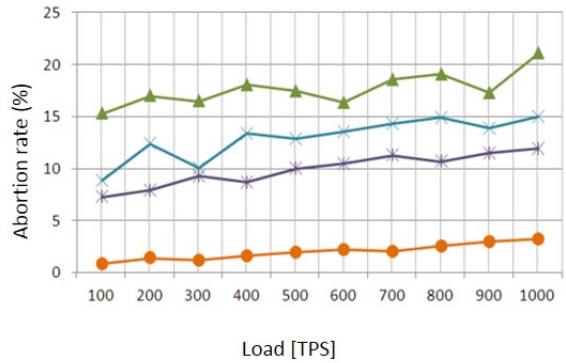
3.1.2 Abortion Rate Results

This metric is important since if we only consider the conclusions derived from the previous section, we will conclude that the lazy protocol is the best replication technique of all proposed. However, if we take a look at how many of these transactions were actually aborted (Figure 9) we can see that it presents the highest abortion rate.

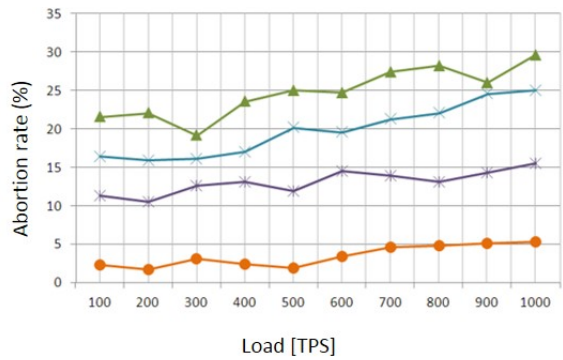
Regarding the other replication techniques, we can see that the weak voting technique presents the best results. This is due to the fact that it holds data items for a long enough period of time and prevents the abortion of other update transactions. However, this conclusion is restricted to our setting. In general, it will depend on the kind of application we are considering. If we have an application that can be split in perfect partitions where no client accesses several partitions, we can take advantage of the lazy technique or, if the percentage of update transactions is low we can use a primary copy replication protocol. Otherwise, the best solution is to go for a weak voting protocol.



a)



b)



c)

Figure 9. Abortion rate for a non hierarchical architecture; a) Write factor: 0.25 b) Write factor: 0.5 c) Write factor: 0.75

3.1.3 System Throughput Results

Finally, we evaluate the system throughput of all replication techniques in a hierarchical architecture with a write factor of 0.5 and 50 % of the read-only transactions accepting old values (see Figure 10). Even when lazy has the best result, as it was explained before it has the highest abort rate, so we highlight the results for the weak voting and certification techniques which are the highest of the rest of replication techniques.

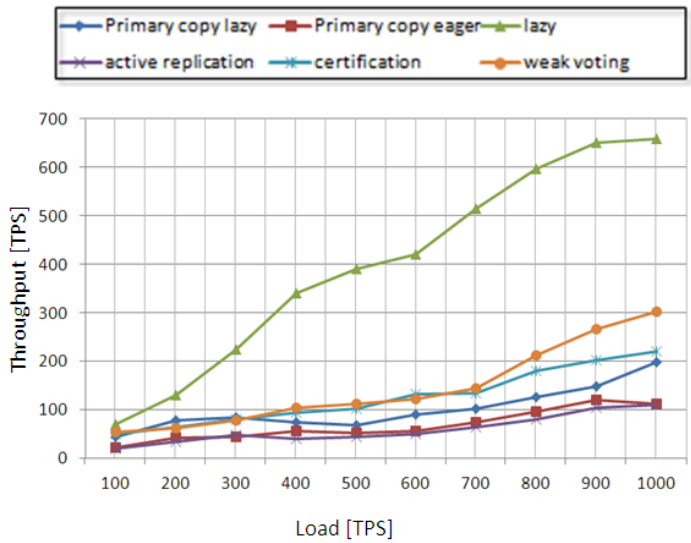


Figure 10. System throughput

4 DISCUSSION

Summing up, we have confirmed that by having a hybrid approach where each partition is replicated through several layers, we can achieve a better response time by mitigating the load that the replicas, participating in the replication technique in the first level, have to afford. However, some replication techniques are affected by the increase of the number of replicas; this is the case of active replication. In this technique the delegate server has to wait for the responses of all its replicas. Moreover, as the replica number increases by promoting the replicas in lower levels, the response time is increased since all they have to do the same work.

The lazy replication technique has the best response time and the highest throughput but also the highest abortion rate. As stated before, this technique makes sense only for non-conflict scenarios which we have not considered. On the

other hand, certification techniques have good results for this hierarchical architecture even if we have a high writing factor. These results were not as good as the weak voting techniques which have the best result in the response times and in the abortion rate. Of course, achieving an optimal performance would require an in-depth study of the system's behavior under different scenarios in order to provide the MM with the necessary knowledge base to take the adequate decisions to adapt configuration parameters to the workload at each time.

5 CONCLUSIONS

We have presented a replicated database system adapted to an elastic cloud environment. In that sense, this system lies in the PaaS category of cloud computing services, more precisely in the category of Database-as-a-Service (DaaS). We have presented a study of well-known transactional database replication techniques in this system. Based on this, we can conclude that there are some areas to explore, specially taking some classic techniques and combining them with new approaches to satisfy the current application demands in terms of scalability and latency.

We have applied the classic database replication techniques to a hierarchical architecture to simulate an elastic environment for a replicated/distributed database to provide a highly scalable and available service. Following the cloud paradigm of pay per use, the proposed system also features an elastic management of resources. This is accomplished by shutting down the database replicas that have not been used and turning them back on only when it is necessary to satisfy the client demands. Also, we have proposed a replication technique based on epidemic updates, which is able to provide different consistency levels according to the requirements of each application and, thus, build a replication hierarchy tree.

The experiments performed using the developed simulator have allowed us to verify that the existence of a hierarchical architecture working with asynchronous updates is able to alleviate the scalability limitations of the traditional replicated database by redirecting transactions that tolerate less recent versions of data to the replicas in the next level.

Acknowledgments

This work has been supported by the Spanish Government under research grant TIN2009-14460-C03 and TIN 2012-37719-C03.

REFERENCES

[1] Wiesmann, M.—Schiper, A.: Comparison of Database Replication Techniques Based on Total Order Broadcast. IEEE Transactions on Knowledge and Data Engineering, Vol. 17, 2005, No. 4, pp. 551–566.

[2] Daudjee, K.—Salem, K.: Lazy Database Replication with Snapshot Isolation. Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06), 2006, pp. 715–726.

[3] Plattner, C.—Alonso, G.—Özsu, T.: Extending DBMSs with Satellite Databases. VLDB Journal, Vol. 17, 2008, pp. 657–682.

[4] Lin, Y.—Kemme, B.—Patiño-Martínez, M.—Jiménez-Peris, R.: Middleware Based Data Replication Providing Snapshot Isolation. Proceedings of SIGMOD, 2005, pp. 419–430.

[5] Das, S.—Agarwal, S.—Agrawal, D.—El Abbadi, Amr: ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud. ACM Transactions on Database Systems, Vol. 38, 2013, No. 1, Art. No. 5.

[6] Kemme, B.—Alonso, G.: A New Approach to Developing and Implementing Eager Database Replication Protocols. ACM Transactions on Database Systems, Vol. 25, 2000, No. 3, pp. 333–379.

[7] Wiesmann, M.: Group Communications and Database Replication: Techniques, Issues and Performance. Ph.D. thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 2002.

[8] Chockler, G. V.—Keidar, I.—Vitenberg, R.: Group Communication Specifications: A Comprehensive Study. ACM Computing Surveys, Vol. 33, 2001, No. 4, pp. 427–469.

[9] Kemme, B.—Alonso, G.: A Suite of Database Replication Protocols Based on Group Communication Primitives. Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS '98), 1998, pp. 156–163.

[10] DeCandia, G.—Hastorun, D.—Jampani, M.—Kakulapati, G.—Lakshman, A.—Pilchin, A.—Sivasubramanian, S.—Vosshall, P.—Vogels, W.: Dynamo: Amazon's Highly Available Key-Value Store. Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07), 2007, pp. 205–220.

[11] Kraska, T.—Hentschel, M.—Alonso, G.—Kossmann, D: Consistency Rationing in the Cloud: Pay Only When It Matters. Proceedings of VLDB Endowment, Vol. 2, 2009, pp. 253–264.

[12] Gong, G.—Liu, J.—Zhang, Q.—Chen, H.—Gong, Z.: The Characteristics of Cloud Computing. Proceedings of 39th International Conference on Parallel Processing Workshops (ICPPW), 2010, pp. 275–279.

[13] Nguyen, B. M.—Tran, V. D.—Hluchý, L.: A Generic Development and Deployment Framework for Cloud Computing and Distributed Applications. Computing and Informatics, Vol. 32, 2013, No. 3, pp. 461–485.

[14] Chang, F.—Dean, J.—Ghemawat, S.—Hsieh, W. C.—Wallach, D. A.—Burrows, M.—Chandra, T.—Fikes, A.—Gruber, R. E.: Bigtable: A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems, Vol. 26, 2008, No. 2, Art. No. 4.

[15] Cooper, B. F.—Ramakrishnan, R.—Srivastava, U.—Silberstein, A.—Bohannon, P.—Jacobsen, H. A.—Puz, N.—Weaver, D.—Yerneni, R.:

PNUTS: Yahoo!'s Hosted Data Serving Platform. Proceedings of the VLDB Endowment, Vol. 1, 2008, No. 2, pp. 1277–1288.

[16] Vaquero, L. M.—Rodero-Merino, L.—Caceres, J.—Lindner, M.: A Break in the Clouds: Towards a Cloud Definition. ACM SIGCOMM Computer Communication Review, Vol. 39, 2009, No. 1, pp. 50–55.

[17] Irún-Briz, L.—Decker, H.—de Juan-Marín, R.—Castro-Company, F.—Armendáriz-Íñigo, J. E.—Muñoz-Escoí, F. D.: MADIS: A Slim Middleware for Database Replication. Euro-Par 2005 Parallel Processing, Proceedings of 11th International Euro-Par Conference, Springer, LNCS, Vol. 3648, 2005, pp. 349–359.

[18] Subramanian, K.—Kandhasamy, P.—Subramanian, S.: A Novel Approach to Extract High Utility Itemsets from Distributed Databases. Computing and Informatics, Vol. 31, 2012, No. 6+, pp. 1597–1615.

[19] Witschel, H. F.: Learning Profiles for Heterogeneous Distributed Information Sources. Computing and Informatics, Vol. 29, 2010, No. 4, pp. 571–584.

[20] Louis-Rodríguez, M. J.—Navarro, J.—Arrieta-Salinas, I.—Azqueta-Alzúaz, A.—Sancho-Asencio, A.—Armendáriz-Íñigo, J. E.: Workload Management For Dynamic Partitioning Schemes in Replicated Databases. Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013), 2013.

Melissa Santana received her M.Sc. degree in information technologies in 2012 from Universidad Pública de Navarra and a Bachelor's degree in computer science from Universidad Central de Venezuela in 2008. Her main research interests are in the area of distributed databases, cloud computing, distributed simulation, data management and GIS databases. Right now, she is working at IMS Health (Brighton, UK).

José Enrique Armendáriz-Iñigo received his Ph.D. degree in 2006 and his M.Sc. degree in telecommunication engineering in 2001 from Universidad Pública de Navarra. He is currently Associate Professor at the same university. His main research interests are in the area of distributed systems, mainly data management in the cloud. Throughout these years, he has participated in several research projects and contributed as a reviewer, PC member or contributor to several international and national conferences as well as in scientific journals.

Francesc D. Muñoz-Escoí received his M.Sc. degree in computer science from Universitat Politècnica de València (UPV) in 1994, and his Ph.D. degree in 2001. He is a member of the Distributed Systems Group of the UPV since 1994 and joined its Instituto Tecnológico de Informática (ITI) since its creation in 1994. He is currently Associate Professor at UPV. His research areas include distributed computing, cloud computing, group communication services, replication protocols, recovery approaches and distributed data management. He has co-authored more than 100 publications in journals and conferences related to these research areas.