2009 28th IEEE International Symposium on Reliable Distributed Systems

On the Cost of Database Clusters Reconfiguration

R. Vilaça J. Pereira R. Oliveira
Computer Science and Technology Center
Universidade do Minho
Braga, Portugal
+351 253 604 430 / +351 253 604 470
{*rmvilaca,jop,rco*}*@di.uminho.pt*

J.E. Armendáriz-Iñigo J.R. González de Mendívil
Departamento de Ingenieria Matematica e Informatica
Universidad Pública de Navarra
Pamplona, Spain
+34 948 168 056 / +34 948 169 521
{*enrique.armendariz,mendivil*}*@unavarra.es*

***Abstract*—Database clusters based on share-nothing replication techniques are currently widely accepted as a practical solution to scalability and availability of the data tier. A key issue when planning such systems is the ability to meet service level agreements when load spikes occur or cluster nodes fail. This translates into the ability to provision and deploy additional nodes.**

Many current research efforts focus on designing autonomic controllers to perform such reconfiguration, tuned to quickly react to system changes and spawn new replicas based on resource usage and performance measurements. In contrast, we are concerned about the inherent impact of deploying an additional node to an online cluster, considering both the time required to finish such an action as well as the impact on resource usage and performance of the cluster as a whole. If noticeable, such impact hinders the practicability of self-management techniques, since it adds an additional dimension that has to be accounted for.

Our approach is to systematically benchmark a number of different reconfiguration scenarios to assess the cost of bringing a new replica online. We consider factors such as: workload characteristics, incremental and parallel recovery, flow control and outdatedness of the recovering replica. As a result, we show that research should be refocused from optimizing the capture and transmition of changes to applying them, which in a realistic setting dominates the cost of the recovery operation.

***Keywords*-Databases; Replication; Group Communication; Performance and QoS; Testing**

I. Introduction

Share-nothing database replication in a cluster of machines has been widely adopted as a solution to increase system throughput and cope with failures. The system involves a set of replicas each locally storing a physical copy of the database. Performance can be increased by appropriately distributing the workload among the replicas and fault tolerance achieved by automatically replacing any failed replica without incurring service outages.

To meet service requirements a reconfiguration of the cluster is often required in order to face load spikes or restore the resilience of the system. Many current research efforts focus on the dynamics of the cluster designing autonomic controllers based on a feedback loop that monitor and react in order to perform such reconfiguration. These systems are usually tuned to quickly react to system changes and spawn new replicas based on resource usage and performance measurements. In contrast, in this paper we are concerned about the inherent impact of deploying an additional database replica to an online cluster. Bringing a database replica online requires updating it to the most current database state while, at the same time, keeping the whole system online. The efficiency of the recovering protocol takes into account both the time required to finish the action as well as the impact on resource usage and performance of the cluster as a whole. If noticeable, such impact hinders the practicability of self-management techniques, since it adds an additional dimension that has to be accounted for.

Recently, a large body of research has been dedicated to the online recovery of replicas in database clusters [13], [2], [9], [10], [23], [16]. These works are all based on a reliable group communication substrate and have in common the use of consistent database replication protocols. Each of these works presents refined techniques to improve the recovery performance aiming at reducing the system's downtime during recovery, the impact on the system's throughput and the time to update a replica and bring it online.

However, it is unfortunate that none of these works provides a detailed evaluation of their techniques under representative workload scenarios. In this paper, our goal is to combine most of the proposed techniques into a streamlined recovery algorithm and systematically benchmark a number of different reconfiguration scenarios to assess the cost of bringing a replica online. This can require the transference of the whole database or just a partial update of a previously failed replica. As a result, we are able to assess their impact in balancing the performance of the recovery process and the overhead imposed to the clustered database service as well as to determine fundamental limits to cluster reconfiguration, discuss the relative merits of different approaches and point out key issues that have to be addressed when implementing such systems.

The paper is organized as follows: Section II describes the replicated database cluster. Section III presents the

Work supported by the Spanish Government under research grant TIN2006-14738-C02-02.

1060-9857/09 $26.00 © 2009 IEEE
DOI 10.1109/SRDS.2009.27

database recovery protocol based on group communication and Section IV its evaluation using different combinations of the protocol's adaptive parameters under two different workloads, TPC-C and TPC-W benchmarks. Section V discusses related work and Section VI concludes the paper.

II. Replicated Database

We consider a fully connected cluster consisting of a set of database replicas hosted by dedicated computers. A database replica may fail by crashing and can afterwards recover. If it recovers it does so with a consistent, possibly outdated, state.

The database replicas interact directly with the clients and coordinate with each other through a replication protocol. The replication protocol depends heavily on a reliable group communication service (GCS) for handling and masking faults (management of the set of active replicas and reliable communication) as well as totally ordering messages within the group. The GCS provides each replica a totally ordered sequence of *views*, each view reflecting the current group of active replicas. Changes in the composition of the group are delivered to the replication protocol through a view change event. A new view differs in the composition of the group in exactly one replica. We assume that GCS ensures view synchrony [5]: two replicas installing two consecutive views deliver the same set of messages between them. Relevant for the correctness of the replication protocol used (with impact on the figures obtained during its evaluation) is the uniformity of all the used communication primitives: a multicast message delivered by a replica (faulty or not) in view v is ensured to be delivered by all non faulty replicas in v.

We are interested in consistent replication protocols. These should provide one-copy equivalence of the centralized consistency criterion at the boundaries of transactions. For the purposes of this study the characteristics of the replication algorithm are not determinant. We opted to use an algorithm with optimistic concurrency control, also known as a certification based algorithm [21], [11]. In contrast to conservative algorithms [20], [1], certification based algorithms allow to extend the centralized consistency criterion (i.e., *first committer wins* [3]) without strengthening it. To keep the replication protocol overhead as low as possible we consider Generalized Snapshot Isolation [6] as the cluster consistency criterion.

We used the Database State Machine [21] protocol with a relaxed certification procedure [6]. The protocol runs as follows. The set of database replicas form a communication group. Each replica holds a full copy of the database and persistently stores the current version of it. Any replica is capable of handling client requests. Clients will execute transactions determined by a load generator that simulates the transaction traffic of a TPC-W [25] and a TPC-C [24]. Both standard workloads represent a good point to compare between intensive memory and disk usage, respectively. Finally, in the case of a replica failure its associated clients are uniformly distributed among the rest of available replicas during the failure period; however, those transactions that were already sent by clients to the failed replica would be aborted.

Once a replica receives a transaction t it executes t without prior coordination with the group. If the transaction successfully reaches the commit phase, then commit is held and t is totally ordered within the group to obtain its commit turn (read-only transactions are directly committed without any interaction with the rest of replicas). The transaction's write set and the database version v_t on which t was executed are multicast. Each replica, once it has processed all transactions ordered before t, *certifies* t by checking its write set against all the items wrote by the sequence of committed transactions applied over v_t. If no conflicts are found t's updates are applied to the local database and committed, otherwise t is aborted.

Together, the agreement on the set of messages delivered to each active replica and the total order provided by the underlying GCS along with a deterministic certification procedure ensure that the whole system acts like a replicated state machine keeping the state of the replicas consistent at the boundaries of transactions. It is worth noting that the updates of a transaction executed remotely are applied in the context of a special *remote transaction*. To ensure the determinism of the whole certification process, remote transactions have high priority and commit despite conflicts with local transactions being executed [17], [18].

III. Recovery Protocol

The recovery of a replica takes place upon the delivery of a new view from the underlying GCS containing an additional replica. The goal of the online recovery protocol is to integrate the replica in the group as quickly as possible while minimizing the impact of the recovery into the database service.

Our online recovery protocol is based on the algorithm presented in [13]. We further enhanced it by adding two other previously proposed techniques meant to improve performance. Namely, the ability to do parallel recovery [10] by having more than one replica contributing with its state and the introduction of *convergence phases* [2], [23]. Our complete algorithm still comprehends basic optimizations such as purging redundant data changes and the compression of the transmitted data.

To help on the recovery of a new replica, each member of the cluster logs every committed write set. This log has the form of $\langle version, writeset \rangle$, where the first element refers to the current database version and the second to the write set itself. The log is maintained in main memory and to limit the size of the log and the volume of the transmitted data, the log is asynchronously purged from redundant entries (updates to

the same tuples of the database). In the worst case, the log can grow to the size of the database itself. The decision to discard part of the log below a given database version, and the copy of the whole database if necessary, can be defined for a given size of the log and number of outdated replicas. An important configuration parameter of the system is the maximum size of the log up to which the recovery protocol outperforms the full copy of a database dump [13], [2], [16].

The recovery protocol runs as follows. Upon the view change introducing a new replica all replicas refrain from sending replication messages. The recovering replica multicast its current version of the database asking for state donors. When this message is delivered, a set of donors is chosen to proceed with the state transfer of the updates missed by the recovering replica according to its announced version. The maximum number of donors is predefined and their selection deterministic (in our current implementation, the donors are the replicas with the lowest id in the group). When using multiple donors the synchronization of the purging of the log may be required. This happens when the donors split the log to transfer in terms of size, e.g. each is responsible for donating 10MB of a 30MB subset of the log. This allows for an even distribution of the load and was the solution we used.

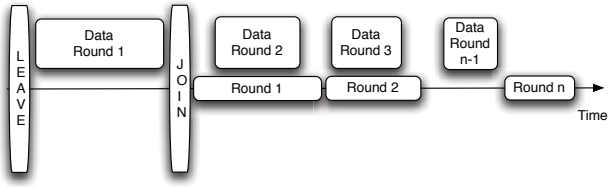


Figure 1. Phases of the recovery protocol

The data transfer is done with a unicast channel and using convergence phases. It has been shown in [23] that recovery time can be substantially reduced if split in convergence phases. During the whole recovery process, the recovering replica does not handle any user requests nor it participates in the replication protocol. On the other hand, once the new view is installed and the state transfer starts all up-to-date replicas resume the database service.

The first convergence phase corresponds to the transference of the log at the time of the view change. Depending on the outdatedness of the recovering replica, this first phase may take a considerable amount of time. Therefore, during this phase, because the system is online the log keeps growing. The log created during the first convergence phase is to be transfered on the second phase. As shown in Fig. 1 this process can be generalized to n phases. This has been shown [23] to outperform the alternative of having the recovering replica buffering the delivered write sets until the missing log is applied. This buffer would then go through the certification process and, when due, the updates applied.

These different convergence phases serve to coordinate the recovery and the replication protocol. The recovery process is said to be about to end when the write set log is below a given threshold or a maximum number of convergence phases is reached. This establishes the synchronization point for the last phase of the recovery process and each donor multicasts a totally ordered "*end recovery*" message. Upon the reception of the first of such messages, the recovering replica starts buffering write sets coming from the replication protocol. These write sets are certified and, possibly, applied after the last convergence phase has been applied. From then on, switching to the replication protocol is straightforward. For the evaluation of the recovery protocol we will therefore take into account different values for the number of state donors and the number of convergence phases as well as the workload during recovery as it directly impacts on the buffering required by the recovering replica after the first convergence phase.

With regard to failures, if the joining replica fails during its recovery then the process is aborted. If it is a donor that fails, the recovering replica restarts the process from the latest database version it knows to be locally consistent.

IV. Evaluation

A. Experimental Setting

Our testing configuration consisted of four computers in a switched Gigabit local area network. This was the minimal setting allowing us to test all the relevant variations of the recovery protocol. We used machines with Intel Core 2 Duo processors running at 2.13GHz, 1GB of RAM and a dedicated SATA hard disk. All machines ran Linux (with kernel version 2.6.22-14-smp). This hardware configuration corresponds to commodity servers and was chosen to be on par with the systems used in recent related work [17], [22], [18], [23], [16].

Each database replica ran an instance of PostgreSQL-G (PostgreSQL 8.1 compliant with the GORDA replication API [4]) and a Java Virtual Machine (1.5.0) running the Escada Replication Server [8]. The Escada Replication Server is split in four components: the *capture component* that communicates with PostgreSQL-G, the *distribution component* that implements the replication protocol described in Section II, the *recovery component* that implements the recovery protocol described in Section 3, and the *apply component* responsible for applying remote transaction updates to each replica. For the group communication substrate the APPIA Group Communication Toolkit [7] has been used.

For our evaluation of the recovery protocol we used our own implementations of the TPC-C [24] and the TPC-W [25] benchmarks. TPC-C is the industry standard online transaction processing benchmark. It mimics a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. The warehouses are hotspots of the system and the benchmark defines 10

client per warehouse. The traffic is a mixture of 8% read-only and 92% update transactions and therefore is used as a write intensive benchmark. On the contrary, TPC-W is read-intensive. It models an Internet commerce environment that resembles real world, business oriented, transactional web applications. It specifies a workload that simulates the activities of an online bookstore with three different consumer patterns that vary the ratio of read-only transactions vs. update transactions: *Browsing Mix* presents the 95% of read-only transaction as opposed to the 5% of update transactions; *Shopping Mix* specifies 80% vs. 20%; and *Ordering Mix* 50% vs. 50%, respectively. TPC-W tends to be processing intensive shifting major resource consumption from storage to CPU. For both TPC-C and TPC-W the size of the database is a function of the desired number of clients. We have run our experiments with a load generator that simulates the workload. As TPC-W benchmark defines a complete 3-tiered architecture, with clients, application server and database servers, that is quite complex to setup and run, our TPC-W load generator rather than mimicking the complete specification replays transaction from a previous generated database execution trace of a full TPC-W benchmark specification. For each experiment, the TPC-W execution trace consisted of 400000 consecutive transactions. Therefore, no client processing overhead is accounted for in the measures that follow; i.e. this process is pretty similar to the one described in [22].

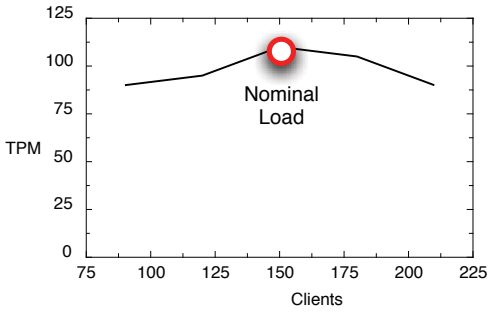
For all tests we chose a workload able to keep all replicas close to their nominal capacity, that is, as busy as possible without saturation of any of resource, as shown in Figure 2 where TPM measures system throughput in transactions per minute. In practice, the TPC-C database has been populated with 15 warehouses which corresponds to a maximum of 150 clients and resulted in a database of 2.2GB in size and, for TPC-W, we chose the *Shopping Mix* configuration with a database populated for 400 clients and 10000 items resulting in a database of 2.4GB. For a higher number of clients the throughput of both benchmarks decreases as the machine's resources get saturated.

Throughout the experiments, no failures occurred during the recovery process and, at anytime, at most one replica was recovering. Each result depicted in Figures 3 and 4 is the average of three independent samples and their standard deviation is negligible.

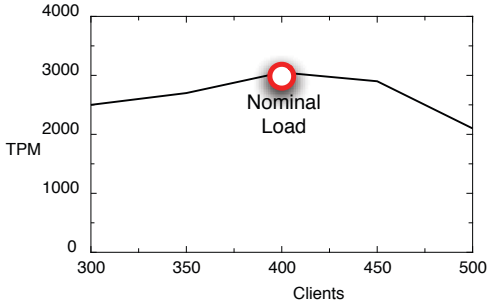
The workflow control was done by limiting the number of clients served by each replica. This was done in the *capture component* of the Escada Replication Server that is notified of all local incoming transactions.

B. Benchmark's Clients

Clients of both load generators ran in dedicated machines of the same network and connected using the JDBC driver. Both benchmark implementations use a simple ad-hoc load balancer that evenly distribute clients per replica. Transactions being handled by a replica that failed are evenly distributed among the live replicas during the failure period but transactions that were already sent by the clients to the failed replica are aborted.



(a) TPC-C



(b) TPC-W

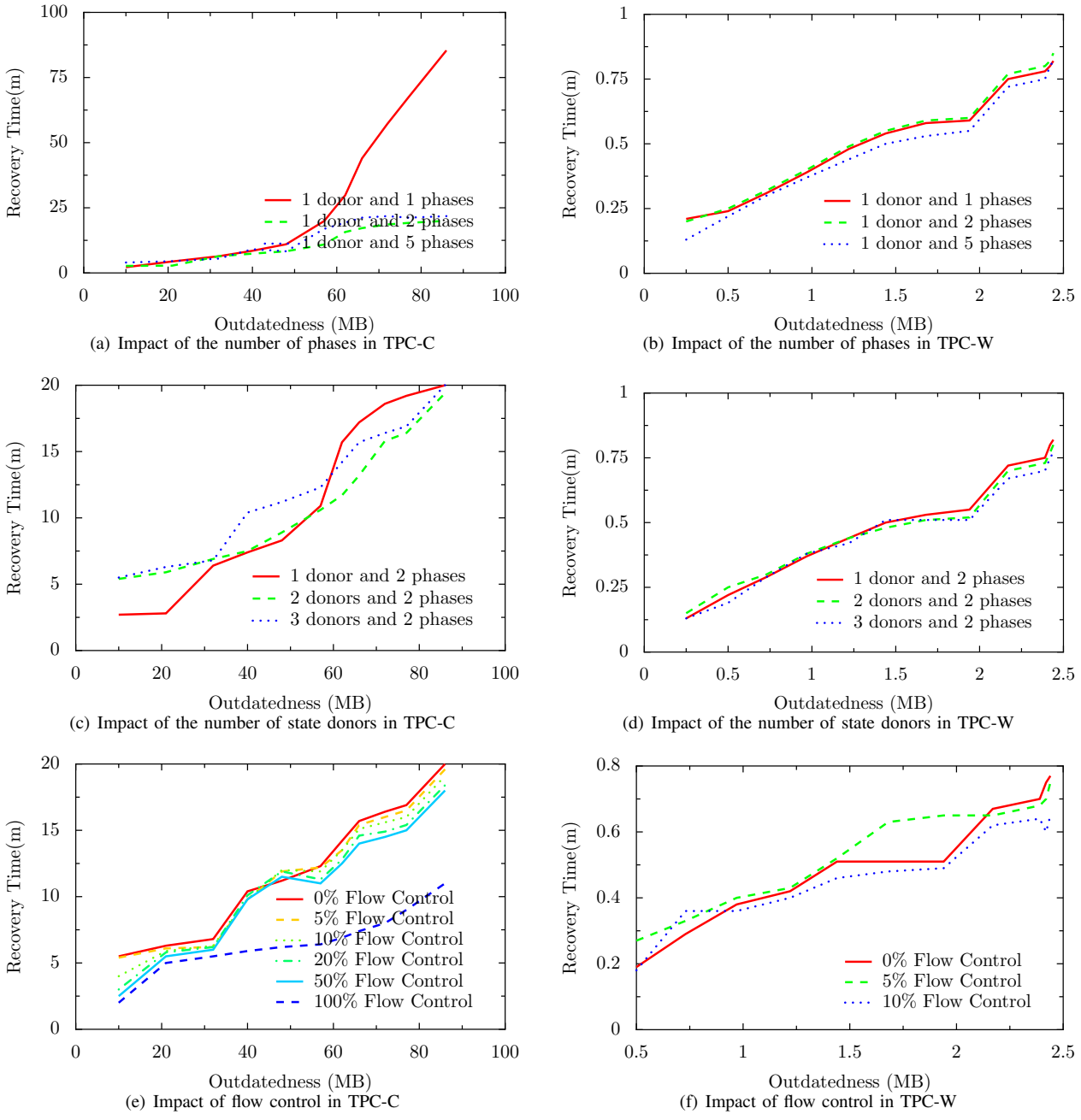
Figure 2. Nominal Load

C. Experiments

In our experiments we looked at three sets of results. First we wanted to measure the impact of three configuration parameters of the recovery protocol: the number of convergence phases, the number of donor replicas and the

Table I
OUTDATEDNESS OF A FAILED REPLICA (MEASURED IN MEGABYTES, MB) AS A FUNCTION OF ITS DOWNTIME (MEASURED IN MINUTES, M)

	Outdatedness (MB)	
Downtime (m)	TPC-C	TPC-W
5	3	0.25
10	10	0.50
15	21	0.73
20	32	0.97
25	40	1.22
30	48	1.44
35	57	1.68
40	62	1.94
45	66	2.17
50	72	2.39
55	77	2.42
60	86	2.44



(a) Impact of the number of phases in TPC-C

(b) Impact of the number of phases in TPC-W

(c) Impact of the number of state donors in TPC-C

(d) Impact of the number of state donors in TPC-W

(e) Impact of flow control in TPC-C

(f) Impact of flow control in TPC-W

Figure 3. Results for TPC-C (left) and TPC-W (right)

flow control of incoming transactions. Then we wanted to measure the overhead of the recovery process as perceived by the users and on the amount of data being handled by the replicas. Finally, we show the time of tasks of replicas.

Each experiment ran as follows. The system was started with all replicas running and then one of the replicas was forced to crash, i.e., by way of killing the PostgreSQL-G and Escada Replication Server processes. The crashed replica was kept offline until the desired outdatedness was reached. The outdatedness of a failed replica is a function of the time the replica is offline and is also dependent on the benchmark as each one has its specific update transactions ratio and write set size, as depicted in Table I. This was given by the size of the write sets of the replication protocol while the replica was down. At that time, the recovery protocol was started. The recovery time, system throughput and log sizes at the different replicas were measured for each of the selected configurations.

1) Recovery Time: The first set of tests relates the recovery time of a replica with its *outdatedness* measured in MB.

Varying the number of convergence phases: In this set of runs we evaluate the impact of using several convergence phases. We present the results of configuring recovery with 1, 2 and 5 convergence phases and using a single replica as state donor. For TPC-C, Figure 3(a) shows that with TPC-C the number of convergence phases is not relevant except when using 1 converge phase. In this scenario, for outdatedness larger than 50 MB, the recovery time has a high increase mainly due to the write intensive workload that makes harder to catch up the rest of available replicas. On the contrary, Figure 3(b) shows that with TPC-W the number of convergence phases is not relevant (notice the different scales of Figures 3(a) and 3(b)).

Varying the number of state donors: The use of parallel recovery using several replicas as state donors has been proposed in [13], [10], [2] as a way to speed up recovery and reduce the overhead at each donor.

For these experiments we have varied the number of state donors from one to three. The recovery protocol has been configured with two convergence phases. The results are depicted in Figures 3(c) and 3(d). For TPC-C, I/O at the recovering replica is the limiting factor. While for small values of outdatedness using a single donor outperforms the added complexity imposed by the synchronization of multiple state donors, for larger values the apparent advantage of multiple donors is diluted by the I/O saturation at the recovering replica. It is worth noting that when recovering, similarly to the replication protocol in use, the application of updates to the replicas is done sequentially. This means that the recovery protocol would not take full advantage of the parallel reception of updates because the bottleneck in the state transference is at the recovering replica.

With TPC-W the differences are again negligible as the amount of updates does not impose any major overhead on the donors at the second convergence phase and, for the three configurations, it is the I/O bandwidth of the recovering replica that dictates the recovery time.

Flow control of incoming transactions: Recovery time may depend heavily on the processing flow during the recovery process. Ultimately, such a throughput could be so large that no replica could ever catch up. In practice however, with full and consistent replication, throughput is determined by the slowest live replica. As such, as long as replicas do not differ enormously in their processing and I/O capacities, ensuring the timely recovery of a replica is perfectly reasonable. With the next experiment we intended to see what was the impact of controlling the systems throughput during recovery on the recovery time of a replica.

We configured the recovery protocol with 2 phases and 3 state donors. Then we imposed a control of 5, 10, 20 and 50% to the flow of incoming transactions with the TPC-C workload to see the impact in the recovery time. We considered also the limit cases: no flow control (as in all previous tests) and an offline system. Figure 3(e) shows the results obtained. It can be seen that reducing, at least, up to 50% of the workload the differences are minimal and mostly due to variance.

With TPC-W, Figure 3(f), the impact is also negligible and again diluted by variance.

2) Impact of Recovery on the System's Throughput: We now show the impact of the recovery of a replica on the system's throughput. We analyze the system throughput (measured in Transactions Per Minute, TPM) under all the previous recovery scenarios with a downtime of 60 minutes. Figures 4(a) and 4(b) depict the results for TPC-C and TPC-W, respectively. For each configuration (1d-2p-0%fc stands for a configuration with 1 state donor, 2 convergence phases and no flow control) we present the system's throughput before and during the recovery process.

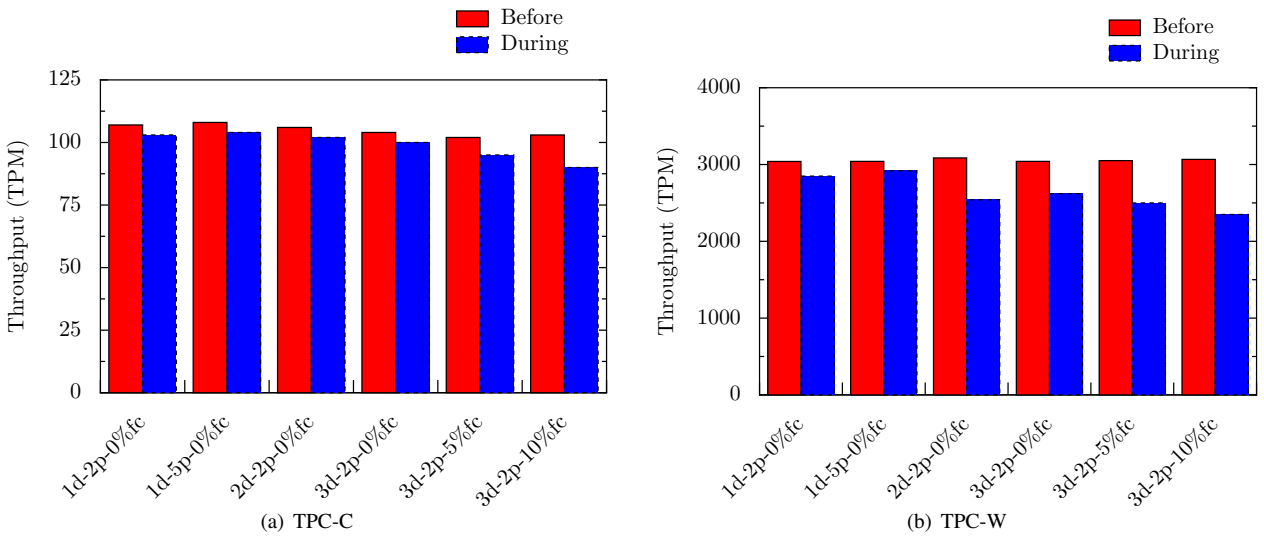
In general, we conclude that throughput is reduced by approximately 15% for the TPC-C workload during the recovery process, and approximately 10% for the TPC-W workload. This impact on system's throughput is not only related to the overhead of the recovery process imposed to donors but also to the cost of GCS view change. It is also interesting that despite the large differences seen so far when comparing the different recovery scenarios, the impact of recovery on the system's throughput is comparable for the two workloads.

3) Evolution of Log Sizes During Recovery: In this section we analyze the amount of data being handled by the replication and the recovery protocols. The results are depicted in Figure 5 for both TPC-C and TPC-W. The database log size was sampled every second during the period of the experiment. We measured the log size in different replicas: in an active replica before recovery started (*Before*); in a state donor after the start of the recovery (*After*); and, in the recovering replica (*Recovering*). Additionally, we measure the size of transferred data from a state donor to the recovering replica (*Transferred*); in the case of multiple donors, one of them was chosen to show these values since they are almost identical and the graphics overlap.

The interesting information conveyed by these figures is the fast transfer between the donor(s) and the recovering replica regarding the log existing before the recovery starts and then the much slower convergence for the the set of pending transactions, i.e. those transactions that need to be certified and applied (*catch-up* process).

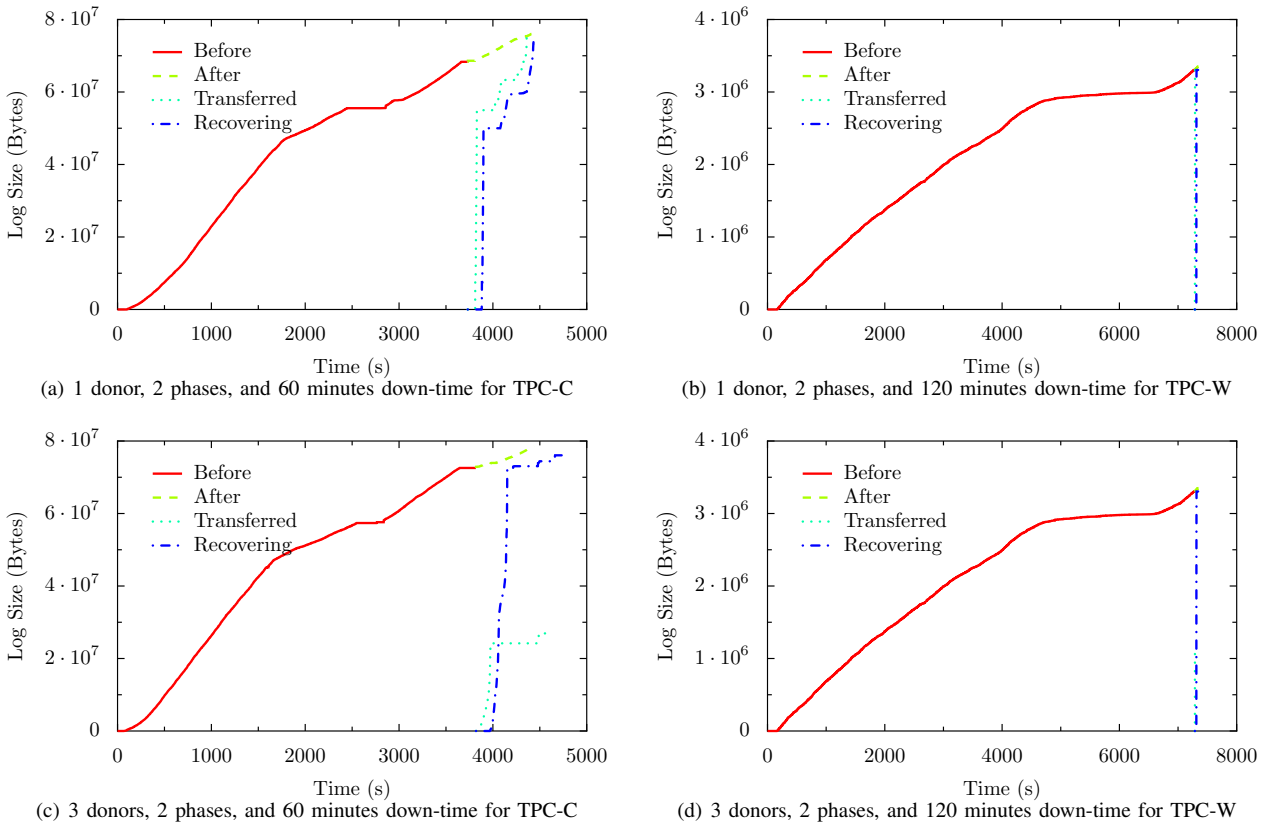
It is also clear that processing of user transactions continues independently of the recovery process; nevertheless, it can be seen that the performance is affected by the increasing slope of the log size during the recovery process.

4) Discussion: Taken as a whole, our results show that there is no clear impact of the different optimizations that have been the focus of recent research efforts. The reason



(a) TPC-C (b) TPC-W

Figure 4. Throughput of user transactions before and during the recovery process



(a) 1 donor, 2 phases, and 60 minutes down-time for TPC-C

(b) 1 donor, 2 phases, and 120 minutes down-time for TPC-W

(c) 3 donors, 2 phases, and 60 minutes down-time for TPC-C

(d) 3 donors, 2 phases, and 120 minutes down-time for TPC-W

Figure 5. Data Log size for TPC-C and TPC-W

for this is that the time for the state donor to read the log and send updates to the network is a mere 15% of the time that the recovering replica needs to read the updates from the network and apply them. In detail, for TPC-C with a downtime of 60 minutes, 2 convergence phases, and 1 state donor, the donor needs 168 seconds to complete its task while the recovering replica needs 1145 seconds. From these, more than 93% (1073 seconds) correspond to the

apply process.

Since the major bottleneck is in applying updates, the recovery time is mainly dictated by the I/O bandwidth of the recovering replica. Therefore, most research has been targeted at optimizing the operations that aren't, by a large margin, limiting factors in overall performance.

V. Related Work

Most of the replication protocols proposed in the literature overlook and do not discuss at all the recovery of failed sites or the addition of new ones (e.g. [6], [12], [14], [17]). On the other hand, those that do cover it either do so by simply presenting the algorithms and informally describing them [2], [9], [10], [13] or are evaluated with ad-hoc workloads [16], [23].

In this section, we compare our work with the recovery solutions proposed in [2], [9], [10], [13], [16], [23]

Like ours, these proposals depart from the work in [13] where several recovery techniques are proposed. The main adopted technique is the lazy data transfer where missed updates are transferred in several phases. Implicitly, it considers these features: first of all, the existence of a threshold that determines whether it is better to perform the full database transfer or only the missed write sets; and, transferring only the latest data item version instead of transferring several times the same data item. These are the ideas that we have included in the implementation of the current recovery protocol and is basically a compilation of what has been described in [10], [2], [9].

An interesting non-blocking recovery protocol is presented in [10] where transaction patterns are known in advance (e.g. stored procedures) and define conflict classes. The recovery process is done per conflict class and, hence, an unaltered partition remains unaffected. The end of the recovery process is performed in two phases. Our recovery protocol does not follow this conflict class philosophy because it is highly application dependent and force transactions to use a pattern that restricts the kind of statements to be executed. On the other hand, our protocol allows transactions to continue their execution even while recovery takes place and permits more degrees of freedom over the number of donors or phases. To the best of our knowledge, the performance of the recovery protocol proposed in [10] has never been evaluated.

The recovery techniques presented in [9] are based on exploiting the ideas of [13] to 1-copy-serializable replicated database systems based on total-order multicast while [2] ports [13] to replicated databases with snapshot isolation replicas. The protocol presented in [2] has been implemented and evaluated in [16], [23]. In [23] a cluster of 4 replicas where only one replica may fail was considered and the evaluation carried out using an ad-hoc benchmark that uses a single table database schema with several combinations of outdatedness and workload. In [16], the same algorithm has been evaluated using 2 replicas that execute the *browsing mix* (80% of read-only transactions) of the OSDL-DBT-1 benchmark [19]. In this work we have used two well-known and widely used standard benchmarks (TPC-C and TPC-W) to evaluate the performance of the recovery protocol with 4 replicas.

VI. Conclusions

With this work we aimed at assessing the impact on performance of bringing a database up-to-date when adding a new replica to a strongly consistent replicated database. Preparing a new replica and bringing it online without stopping the replicated database is not a straightforward task. The joining node needs to become part of the coordinated computation, to be updated to the most current state of the system and start to handle user requests with minimal impact on the performance of the system.

As part of our database replication framework [8] we implemented a database recovery protocol that combines several techniques proposed in the literature [13], [2], [9], [10] aimed at speeding up the process. These techniques had not been evaluated in a practical setting under representative workloads and it was our intention to discover to what extend their use would allow a system administrator to find the desired balance between the shortest recovery period and the impact on the system's performance.

We used a cluster of commodity servers reflecting a common balance between processor, memory, storage and network resources as the typical target system for small to medium size clustered databases. Much to our surprise, the results of all of our tests did not reveal any relevant effect of the techniques incorporated by our protocol in the recovery time of the replicas or impact on the overall cluster performance. The reason for this was that, in our setting, the capacity of the recovering replica to apply the received state turns out to be the salient limiting factor. Since most of the recovery protocol enhancements under evaluation aim at quickly *feed* the recovering replica (while minimizing the impact of the overall system response) its inability to timely process the updates defeats their purpose.

To streamline the whole recovery process and quickly bring a new replica online the strong optimization of the *apply* process is essential. This cannot be simply regarded as a matter of increasing storage bandwidth through the use of striping techniques or the addition of faster hard disks. While such an improvement could definitively help mitigating the problem in our testbed it would not keep up with a compatible investment on processing capabilities. A major improvement could be achieved by keeping a binary log for recovery that could be efficiently injected into the recovery replica without going through all the transaction processing pipeline. However, the direct cost of such an approach would be the need to work inside the database engine and to compromise the heterogeneity of the system.

It seems clear to us that the key solution for the problem in hand is two-fold: the parallelization of the apply process inside the replication protocol and the use of on-demand state transfer [15].

REFERENCES

[1] Y. Amir and C. Tutu. From total order to database replication. In *In Proc. of Int. Conf. on Distr. Comp. Systems (ICDCS*, pages 494–503. IEEE, 2002.

[2] J. E. Armendáriz-Íñigo, F. D. Muñoz-Escoí, J. R. Juárez-Rodríguez, J. R. González de Mendívil, and B. Kemme. A recovery protocol for middleware replicated databases providing GSI. In *ARES*, pages 85–92. IEEE-CS, 2007.

[3] H. Berenson, P. A. Bernstein, J. Gray, J. Melton, E. J. O'Neil, and P. E. O'Neil. A critique of ANSI SQL isolation levels. In M. J. Carey and D. A. Schneider, editors, *SIGMOD Conference*, pages 1–10. ACM Press, 1995.

[4] N. Carvalho, A. Correia Jr., J. Pereira, L. Rodrigues, R. Oliveira, and S. Guedes. On the use of a reflective architecture to augment database management systems. *Journal of Universal Computer Science*, 13(8):1110–1135, 2007.

[5] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[6] S. Elnikety, F. Pedone, and W. Zwaenopoel. Database replication using generalized snapshot isolation. In *SRDS*. IEEE Computer Society, 2005.

[7] FCUL. APPIA communication framework. Accessible in URL: http://appia.di.fc.ul.pt/, 2007.

[8] GORDA. ESCADA replication server. Accessible in URL: http://escada.sourceforge.net/, 2007.

[9] J. Holliday. Replicated database recovery using multicast communication. In *NCA*, pages 104–107. IEEE Computer Society, 2001.

[10] R. Jiménez-Peris, M. Patiño-Martínez, and G. Alonso. Non-intrusive, parallel recovery of replicated data. In *SRDS*, pages 150–159. IEEE Computer Society, 2002.

[11] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors, *VLDB*, pages 134–143. Morgan Kaufmann, 2000.

[12] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[13] B. Kemme, A. Bartoli, and Özalp Babaoglu. Online reconfiguration in replicated databases based on group communication. In *DSN '01: Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*, pages 117–130, Washington, DC, USA, 2001. IEEE Computer Society.

[14] B. Kemme, F. Pedone, G. Alonso, A. Schiper, and M. Wiesmann. Using optimistic atomic broadcast in transaction processing systems. *IEEE Trans. Knowl. Data Eng.*, 15(4):1018–1032, 2003.

[15] H. A. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *3rd European Conference on Computer Systems (Eurosys)*, Nuremberg, Germany, April 2009.

[16] W. Liang and B. Kemme. Online recovery in cluster databases. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 121–132, New York, NY, USA, 2008. ACM.

[17] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *SIGMOD Conference*, 2005.

[18] F. D. Muñoz-Escoí, J. Pla-Civera, M. I. Ruiz-Fuertes, L. Irún-Briz, H. Decker, J. E. Armendáriz-Iñigo, and J. R. González de Mendívil. Managing transaction conflicts in middleware-based database replication architectures. In *SRDS*, pages 401–410, 2006.

[19] Open Source Database Lab. OSDL database test 1. OSDL-DBT-1. Accessible in URL: http://www.sourceforge.net/projects/osdldbt/, 2002.

[20] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. MIDDLE-R: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4):375–423, 2005.

[21] F. Pedone. *The database state machine and group communication issues (Thèse N. 2090)*. PhD thesis, École Polytecnique Fédérale de Lausanne, Lausanne, Switzerland, 1999.

[22] C. Plattner, G. Alonso, and M. T. Özsu. Extending dbmss with satellite databases. *VLDB J.*, 17(4):657–682, 2008.

[23] M. I. Ruiz-Fuertes, J. Pla-Civera, J. E. Armendáriz-Iñigo, J. R. González de Mendívil, and F. D. Muñoz-Escoí. Revisiting certification-based replicated database recovery. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 489–504. Springer, 2007.

[24] TPC. TPC benchmark C. Standard Specification. Accessible in URL: http://www.tpc.org/tpcc/, 2007.

[25] TPC. TPC benchmark W. Standard Specification. Accessible in URL: http://www.tpc.org/tpcw/, 2007.