Recovery in CloudDBAppliance's High-availability Middleware

Hugo Abreu[1,2], Luis Ferreira[1,2], Fábio Coelho[1,2], Ana Nunes Alonso[1,2] and José Pereira[1,2]
[1]*INESC TEC, Porto, Portugal*
[2]*Universidade do Minho, Braga, Portugal*
{*hugo.m.abreu, luis.m.ferreira, fabio.a.coelho, ana.n.alonso*}*@inesctec.pt, jop@di.uminho.pt*

Keywords: CloudDBAppliance, High-availability, Recovery.

Abstract: In the context of the CloudDBAppliance (CDBA) project, fault tolerance and high-availability are provided in layers: within each appliance, within a data centre and between datacentres. This paper presents the recovery mechanisms in place to fulfill the provision of high-availability within a datacentre. The recovery mechanism takes advantage of CDBA's in-middleware replication mechanism to bring failed replicas up-to-date. Along with the description of different variants of the recovery mechanism, this paper provides their comparative evaluation, focusing on the time it takes to recover a failed replica and how the recovery process impacts throughput.

1 INTRODUCTION

CloudDBAppliance aims at delivering a database appliance (software and hardware), leveraging the capabilities of newer hardware by leveraging NUMA awareness for terabyte-scale in-memory processing with high-availability. High-availability requires a recovery mechanism for failed or even new replicas. This paper focuses on the intra-datacentre recovery mechanism for CloudDBAppliance. Simply put, the goal is to ensure that if an appliance fails, a standby is able to take over, powering a transparent execution scenario for the client, thus hiding faults and maintaining the perceived quality of service. The operational database is the fulcrum of the high-availability effort, as it holds the data that needs to be persisted. The state of other components such as of the in-memory many-core analytics framework (also a part of the project) is considered volatile and can be straightforwardly rebuilt from the data in the operational database as needed, in the event of a failure.

In the context of the CDBA project, ensuring high availability within a datacentre requires a replication mechanism that, in case of failure, enables the failover of the operational database to a consistent and up-to-date standby replica, running in a different appliance. There are a number of different models for database replication that mainly differ on: whether transactions are executed at each replica, or just at one while others apply updates; and how (if at all) replicas are allowed to diverge.

Recovery and replication capabilities work in tandem to provide high-availability: the replication mechanism ensures there are multiple consistent replicas of the data so that the system tolerates the loss of some and the recovery mechanism makes it possible to bring failed or clean replicas up to a consistent state.

In this paper we present the architecture of the recovery mechanism for CloudDBAppliance along with the motivating design constraints. Section 2 covers background on recovery and fault-tolerance mechanisms. Section 3 introduces the overall architecture for the recovery and replication middleware, with Section 4 showing some results on the selected recovery approaches. Section 5 concludes the paper and overviews the major takeaways.

2 BACKGROUND

Recovery, as discussed throughout this paper, refers to the process of bringing failed or clean replicas up-to-date, so that the set of available replicas is consistent according to some criteria. Transactional recovery, in the sense of recovering a single database from an inconsistent (as in the C in ACID) state due to, for example, a transaction being aborted, is out of the scope of this discussion.

How to perform the recovery of failed replicas or how to integrate clean replicas into a highly available system should be a concern when designing replication mechanisms, even if this is often not the case (Vilaca et al., 2009) in the literature (e.g., (Elnikety et al., 2005; Kemme and Alonso, 2000)). However, CDBA's goal of providing highly available database appliances, requires a suitable recovery mechanism to be designed and implemented.

The replication approach (active vs passive) and whether the replication protocol is synchronous or asynchronous have a considerable impact on the design of the recovery protocol, namely to determine when a recovered replica can be considered to be available. For example, asynchronous replication protocols may accept a level of outdatedness in some replicas which may enable less costly recovery strategies or allow recovered replicas to be considered available sooner without disrupting the expected consistency level.

An important design decision is whether recovery is to be done online (Kemme et al., 2001) or offline (Amir, 1995). Offline recovery means that the system somehow becomes unavailable whenever a replica needs to be recovered. Online recovery, on the other hand, means that the system remains available during a recovery process. Again, CDBA's goal of providing highly available database appliances requires recovery to be done online.

The main goal of the recovery process requires some kind of state transfer to the recovering replica. Several approaches can be undertaken ranging from transferring state in bulk or using an incremental approach, from a single or multiple donors. These approaches define trade-offs on how quickly a replica is recovered and the impact on the performance of the overall system. Different approaches may be selected based on a multitude of factors: how far behind the recovering replica is; whether its state is consistent, even if outdated; the number of available donors, etc. The impact of these factors on recovery has been evaluated and analysed in (Vilaca et al., 2009). It may also be possible to take advantage of workload patterns and/or data partitioning to improve recovery, namely when applying missing updates to a recovering replica (Jiménez-Peris et al., 2002).

Intra-datacentre recovery, in the context of CDBA, presents different challenges from the target models generally considered when designing recovery protocols. The main differences are: the substantial computing power of each replica, due to the use of the state-of-the-art, many-core Bullion hardware, as opposed to considering commodity or COTS servers; a very low-latency, high-bandwidth communication network connecting replicas, as opposed to higher latency LAN or WAN networks; and the number of available replicas, which is restricted to two, the minimum to provide high-availability, as opposed to quorum based solutions, which require a minimum of 3 replicas. These differences have a fundamental impact on the design of the replication mechanism and consequently, the recovery mechanism, as these open up the possibility of exploring unusual trade-offs on throughput and availability.

2.1 Replication in CloudDBAppliance

Due to the interdependence of the replication and recovery mechanisms, this section presents an overview of CDBA's intra-datacentre replication protocol. Further details can be found in (Ferreira et al., 2019).

The recovery mechanisms considered were designed and implemented as part of the replication middleware layer that provides isolation between users and the underlying operational database. To achieve this, the middleware layer is placed as a top tier layer, intercepting SQL statements and performing all required steps to accommodate the replication mechanisms. The replication middleware removes non-determinism from requests and ensures these are totally-ordered. The middleware approach simplifies integration and extends the possibility of considering this replication mechanism beyond the CloudDBAppliance project, by offering a completely decoupled solution. This can be done by embedding the middleware in the JDBC driver used for communication between clients and database servers.

The replication middleware relies on a JDBC-enabled API, that contains key interfaces between a client proxy and server side stub. Key interfaces were selected by reusing V-JDBC for JDBC request interception and hand-over between client proxies and server. As detailed and assessed in (Ferreira et al., 2019), choosing V-JDBC allowed for a flexible environment where the transport protocol can be customised according to the application itself.

The replication architecture, depicted in Figure 1 is based on a set of reliable distributed logs. These enable decoupling clients from the replication manager instances and the operational databases. The log allows requests to be stored reliably and with total order guarantees. Briefly, when a client application sends a SQL request, the request is sent to a write-proxy that acts as a handler for the distributed log structure. After being reliably stored, the distributed log structure is used by the replication manager instances that pull the requests and push these for execution at its local operational database instance. The distributed log, conceptually considered as part of the

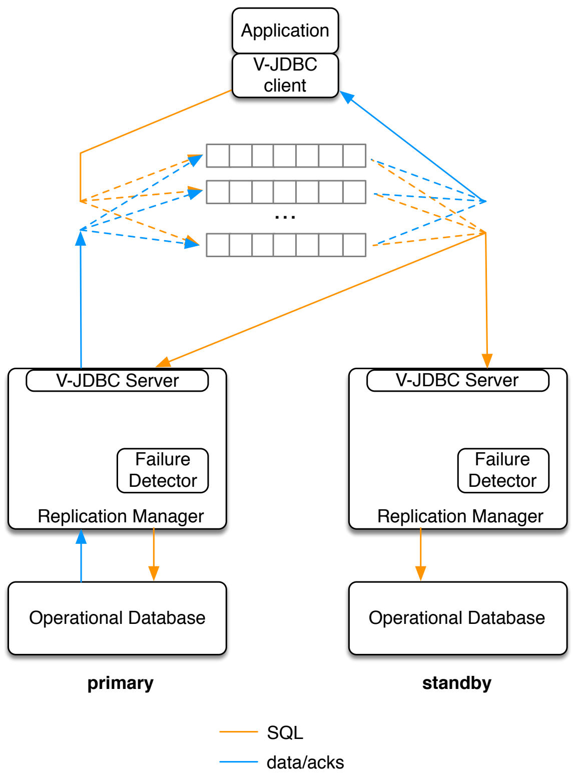

Figure 1: Replication in CloudDBAppliance.

replication mechanism, allows for a customisable deployment, making it possible to launch as many instances as required. Storing requests reliably in the distributed log makes it possible to keep processing in the critical write path to a minimum and, therefore, minimizing the performance impact of the replication middleware on the operational database. Additionally, read and write paths are decoupled.

Also, judiciously partitioning requests among different logs, makes it possible to create independent request queues, establishing distinct shards, and thus enabling requests from different shards to be processed in parallel leading to performance gains.

3 FAILOVER AND RECOVERY FOR HIGH AVAILABILITY

The recovery mechanism and the replication mechanism overviewed in Section 2 are complementary.

The failover and recovery mechanisms depends on the replication manager's failure detector component. Briefly, the failure detector exchanges heartbeats through a reliable communication channel and monitors that channel for the presence of heartbeats from the other replication manager instance. The absence of heartbeats from a given replication manager instance triggers recovery and/or failover procedures as appropriate: a failed primary replica triggers the failover procedure; as a failed replica reboots and reintegrates the system, a recovery process takes place before it is able to enter the standby state.

Next, a description of the recovery process is provided.

1. The replication manager requests a state transfer from the live appliance to the restarted replica, in order to bring it up to a consistent state.
2. The transferred state is installed in the recovering replica.
3. The recovering replica checks the distributed log for outstanding requests and determines the point from which it can enter the standby state.
4. The recovering replica reaches the designated outdatedness point and enters the standby state, ending the recovery process.

Because replication is asynchronous, the point of outdatedness at which the recovering replica can enter the standby state can be configurable. One possibility is to allow the recovering replica to enter the standby state as soon as step 3 is performed.

In fact, different requirements can be defined to end the recovery process depending on whether it was the standby or the primary that failed. The failover mechanism additionally requires the standby replica to determine from the log the point (state) from which it can start answering clients assuming the role of primary. Missing updates are reconstructed from the information in the distributed log. Thus, both the intra-datacentre failover and recovery mechanisms take advantage of the persistence of the distributed log.

Because of the decoupling between clients and replicas, neither recovery nor failover are visible, in general, to client applications. This property depends on the ability of the recovering replica to catch up. The low-latency, high-bandwidth network connection and the high computing capacity of the Bullion are instrumental to providing it. Results in Section 4.2 support this ability.

4 EVALUATION

The evaluation for the recovery process of the intra-datacentre high-availability mechanisms assesses the time to identify, request, transfer and reintegrate state on the new or faulty replica. The most relevant metric to observe is the Mean Time To Recovery (MTTR). One major factor of the MTTR metric is expected to

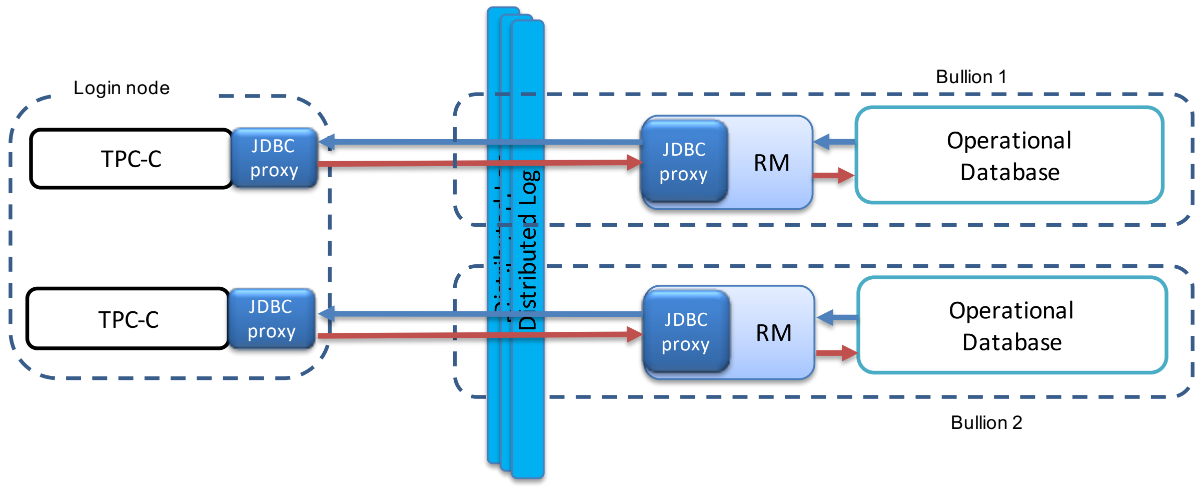

Figure 2: Configuration for the evaluation campaign.

be the amount of data to transferred, which depends on overall database size and the type of failure. For example, if the failed replica has consistent data snapshots available, it may only need to pull and synchronise over any missing snapshots. Otherwise, recovering the replica requires a full state transfer. However, remote file sharing protocols can be used to complement and improve data transfer. We focus on techniques for state transfer: full state transfer using a backup mechanism and using rsync to minimize the transferred data. The impact of the recovery process on transactional activity is also analysed.

4.1 Benchmarking Campaign

Experiments were conducted in the scenario depicted in Figure 2. Each appliance is a standalone state-of-the-art Bullion machine, tailored for CloudDBAppliance.

The hardware specification for the machines considered is detailed in Table 1.

The industry-standard TPC-C benchmark was used to induce a transactional load on the system. The TPC-C specification models a real-world scenario where a company, comprised of several warehouses and districts, processes orders placed by clients. The workload is defined over 9 tables operated by a transaction mix comprised of five different transactions, namely: New Order, Payment, Order Status, Delivery and Stock-Level. Each transaction is composed of several read and update operations, where 92% are update operations, which characterizes this as a write heavy workload. The benchmark is divided into a load and an execution stage. During the first stage, the database tables are populated and during the second stage, the transaction mix is executed over that dataset. TPC-C defines how these tables are populated and also defines their size and scaling requirements, which is indexed to the number of configured warehouses in the system. The outcome of this benchmark is a metric defined as the maximum qualified throughput of the system, tpmC, or the number of New Order transactions per minute.

In order to evaluate the scalability of the recovery mechanism, the TPC-C benchmark was in two load configurations, featuring 100 and 1000 warehouses.

In these experiments, execution starts with two live appliances executing the requests of the running benchmark. At some point, a fault is induced. Because our focus is on evaluating the impact of different techniques for state transfer, in these experiments, recovery is manually triggered. This makes it possible to evaluate different recovery scenarios, inducing different degrees of divergence between the live replica and the failed replica. As such, we measured the MTTR from the time the recovery process is triggered until the state has been installed on the replica.

4.2 Results

Figure 3 reports on the MTTR and the impact of recovering a failed standby on the benchmark's throughput using the default backup process. Using the default backup process creates a full copy of all data handled by a replica. This process will create a base copy, incrementally expanded as the database grows. A recovery mechanism using the default backup process must do a full state transfer in the event of a failure.

The recovery process was triggered 20 seconds after the replica failed. As the failed replica tries to resume activity, its replication manager requests a

Table 1: Hardware description for the evaluation campaign.

Machine	Login node	Bullion 1	Bullion 2
CPU Vendor	Intel	Intel	Intel
CPU Model	Xeon E5-2667 v3 3.2GHz	Xeon Platinum 8150 3.00GHz	Xeon Platinum 8153 2.00GHz
CPU Number	16	192	256
Memory	65.69GB	4.2 TB	3.16 TB
Storage type	SSD	SSD (shared)	
Storage capacity	250 GB	4.9 TB (shared)	

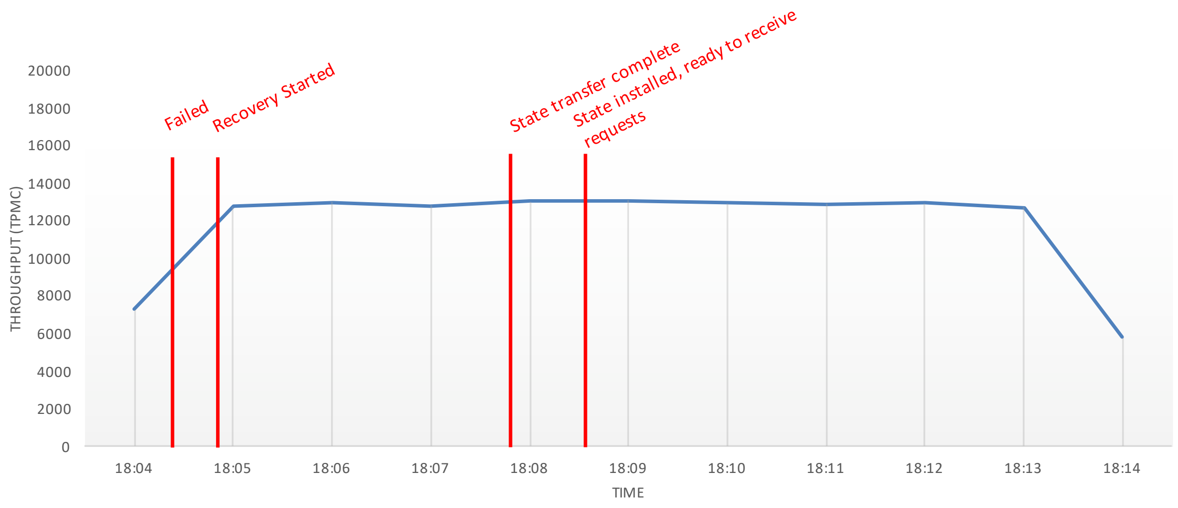

Figure 3: Recovery using the default backup process for 100 warehouses.

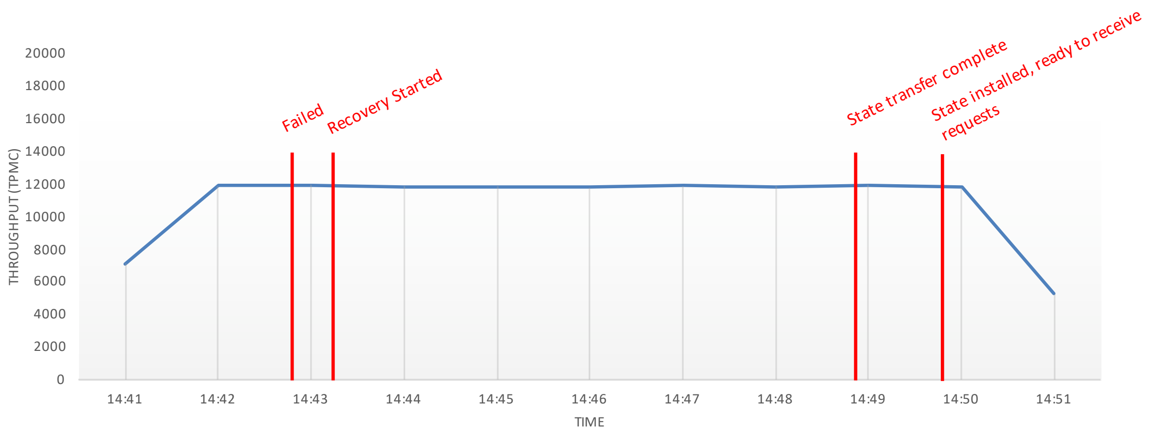

Figure 4: Recovery using rsync for 100 warehouses.

state transfer from the live replica, which completed during the following 3 minutes. After installing the transferred state, the replica determines the last (globally identified) transaction to be processed and then resumes normal transaction execution from the distributed log. Overall, the Mean Time To Recovery was approximately 4 minutes. During the entire recovery process, the observed throughput did not suffer a significant impact, as depicted by the blue line in Figure 3

Figure 4 showcases the same configuration as in Figure 3, but the rsync protocol is used instead to bring the recovering database instance up-to-date with the donor. The rationale is that using rsync may avoid a full state transfer. Again, overall, the throughput performance during the recovery process did not present a considerable impact. However, using rsync actually caused state transfer to take longer, over 5 minutes, actually having a negative impact on the MTTR, which increased to just below 7 minutes. Because the Bullions communicate through a high speed connection, the overhead of having to determine ex-

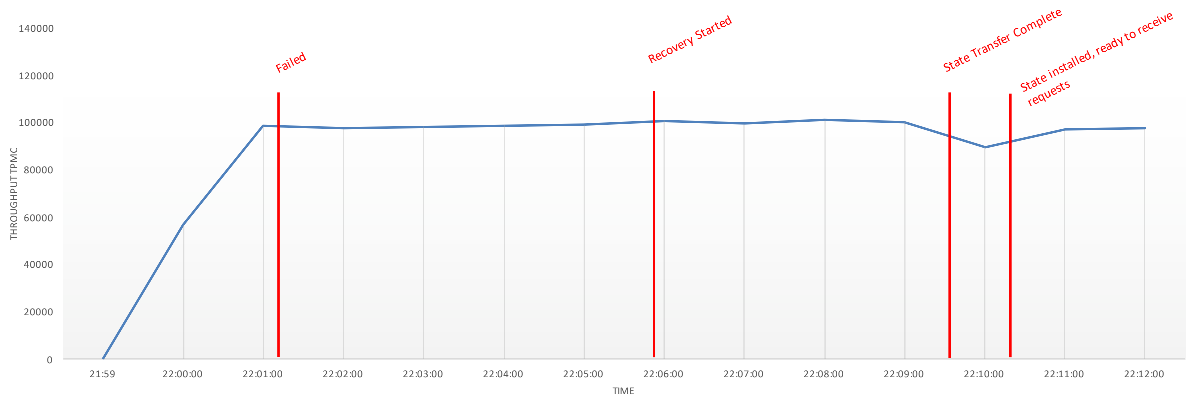

Figure 5: Recovery using the default backup process for 1000 warehouses.

actly which data need to transferred, particularly considering the operational database instance remains online during the process, actually outweighs the impact of sending more data through the network.

Figure 5 depicts the same scenario as Figure 3 but for a configuration with 1000 warehouses. The main goal was to evaluate what the impact on the MTTR would be, when considering a larger dataset. Also, the recovery process was only triggered after a considerable delay (over 5 minutes) to make sure the recovering replica is significantly behind the live replica. While it might be expected for state transfer to take significantly longer than the previous experiment, results show that using the default recovery mechanism is scalable providing that the replicas communicate through a high-speed network connection. Again, regarding throughput, the impact was minimal, as in average the throughput remained stable at around 96 thousand tpmC, even if a small throughput decay was registered during state transfer and installation.

5 CONCLUSION

This paper introduced CloudDBAppliance's intra-datacentre recovery mechanism, essential to its intra-datacentre high-availability framework. It takes advantage of the middleware architecture of the replication mechanism and of the persistence provided by the distributed log. This framework takes advantage of differentiating characteristics of the CloudDBAppliance project, stemming from its state-of-the-art hardware. The recovery process was evaluated using two alternative state transfer techniques. Results show that the impact of the recovery process on a transaction workload induced by running the TPC-C benchmark is negligible. Moreover, results show the recovery mechanism scales well regarding an increased transactional workload and increased divergence of the recovering replica. The proposed replication, failover and recovery mechanisms provide high-availability while still decoupling clients from database instances, enabling its applicability to other scenarios. Regarding future work, we plan to accomdate the inter-datacentre configuration, where the recovery mechanisms need to accomodate updates via the WAN.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Horizon 2020 – The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 732051 and by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalization - COMPETE 2020.

REFERENCES

Amir, Y. (1995). *Replication using group communication over a partitioned network*. PhD thesis, Citeseer.

Elnikety, S., Pedone, F., and Zwaenepoel, W. (2005). Database replication using generalized snapshot isolation. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 73–84. IEEE.

Ferreira, L., Coelho, F., Alonso, N., and Pereira, J. (2019). Towards intra-datacentre high-availability in cloudbappliance. In *CLOSER (1)*.

Jiménez-Peris, R., Patiño-Martínez, M., and Alonso, G. (2002). Non-intrusive, parallel recovery of replicated data. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 150–159. IEEE.

Kemme, B. and Alonso, G. (2000). A new approach to developing and implementing eager database replication protocols. *ACM Transactions on Database Systems (TODS)*, 25(3):333–379.

Kemme, B., Bartoli, A., and Babaoglu, O. (2001). Online reconfiguration in replicated databases based on group communication. In *2001 International Conference on Dependable Systems and Networks*, pages 117–126. IEEE.

Vilaca, R. M. P., Pereira, J. O., Oliveira, R. C., Armendariz-Inigo, J. E., and de Mendivil, J. R. G. (2009). On the cost of database clusters reconfiguration. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 259–267. IEEE.