Towards Intra-Datacentre High-Availability in CloudDBAppliance

Luis Ferreira, Fábio Coelho, Ana Nunes Alonso and José Pereira
INESC TEC & Universidade do Minho, Braga, Portugal

Keywords: CloudDBAppliance, High-Availability, Replication.

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 data centres. This paper presents the proposed replication architecture for providing fault tolerance and high availability within a data centre. This layer configuration, along with specific deployment constraints require a custom replication architecture. In particular, replication must be implemented at the middleware-level, to avoid constraining the backing operational database. This paper is focused on the design of the CDBA Replication Manager along with an evaluation, using micro-benchmarking, of components for the replication middleware. Results show the impact, on both throughput and latency, of the replication mechanisms in place.

1 INTRODUCTION

CloudDBAppliance aims at delivering a database appliance (software and hardware), leveraging the capabilities of newer hardware by incorporating NUMA awareness, terabyte-scale in-memory processing and high-availability. This paper focuses on how to provide a highly available configuration using resources available within a data centre. Simply put, the goal is to guarantee that if an appliance fails, a standby is available 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, either if used directly by an external application or along with an in-memory many-core analytics framework (also a part of the project), providing it with the necessary data.

Replication is often presented as the solution to achieve highly dependable database services. Existing database replication architectural models should then be examined to determine to what extent their assumptions, strengths and weaknesses hold in this type of scenario. Namely, the consistency guarantees that can be afforded, the cost to do so in terms of latency and processing resources, and their implementation cost, should be analysed. Ensuring high availability within a datacentre requires defining a replication algorithm that, in case of a 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.

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

2 BACKGROUND

Replication protocols are often divided into two distinct approaches: active and passive replication. Active replication protocols follow the state-machine approach (Schneider, 1990) where the database is considered to behave like a state-machine in which each operation deterministically causes a change in state: each operation is forwarded to every replica, which then executes it. In order for this approach to be applicable, operations must be guaranteed to be deterministic, precluding the usage of current time values and random numbers, as these would likely differ between replicas.

In contrast, in passive replication protocols, com-

Ferreira, L., Coelho, F., Alonso, A. and Pereira, J.
Towards Intra-Datacentre High-Availability in CloudDBAppliance.
DOI: 10.5220/0007876306350641
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 635-641
ISBN: 978-989-758-365-0

monly referred to as primary-backup, only the primary replica executes the transaction, propagating the transaction's write set to other replicas. The primary's native database engine concurrency control decides which transactions to commit or abort, and in which order. To ensure that replicas remain consistent, these must know or decide on the same serialization order as the primary. In a multi-primary setting, i.e., where, for example, different replicas may have the role of primary for different parts of the data, each transaction still executes in a single primary, but having several primaries means that these must agree on a total order for transaction execution, as a transaction might update data owned by multiple primaries. If replicas apply updates according to that total order, strong consistency is guaranteed. Group communication protocols that guarantee message delivery with appropriate semantics, which are instances of the abstract consensus problem (Guerraoui and Schiper, 2001), can be used for that purpose. In (Wiesmann et al., 2000), the authors compare different approaches for replication as well as the primitives needed in each case and a survey of atomic broadcast algorithms can be found in (Défago et al., 2004).

Because the total order property guarantees that all replicas receive the same set of messages and that messages are delivered in the same order to all replicas, the transaction order can be established simply by sending the transaction identifier (along with other relevant metadata) to the group; if transactions are queued in the same order in which the respective messages are delivered, the queues at each replica will be identical and can be considered as instances of a replicated queue. Because active replication requires every replica to execute every transaction, if non-determinism in transactions is allowed and strongly consistent replication is a requirement, performance is limited by the slowest replica in the group. While passive replication protocols do not suffer from this limitation, transferring large write sets across the network to several replicas can be costly. Protocols that combine active and passive replication have been proposed (Correia Jr et al., 2007). There have also been proposals for mitigating the limitations of state-machine replication, namely by implementing speculative execution and state-partitioning (akin to partial replication) (Marandi et al., 2011) and eschewing non-determinism by restricting valid execution to a single predetermined serial execution (Thomson and Abadi, 2010). Using primary-backup (and multi-primary), ownership of data partitions must be guaranteed to be exclusive. This means that when the primary fails, the database must block until a new primary is found, usually through a leader election protocol. This is costly, particularly in churn-prone environments.

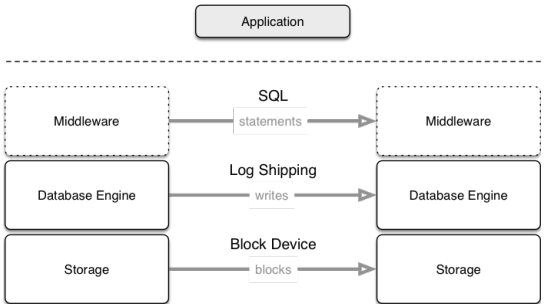

Figure 1: Strategies for database replication.

Having stand-by failover replicas might avoid most runs of the leader election protocol, but at the cost of increasing the number of replicas that need to be updated in each transaction, thereby increasing network utilization and generally increasing the number of nodes in the system without a corresponding improvement in system throughput. Update-everywhere protocols avoid this issue because all replicas are equivalent. Again, replicas must apply updates according to the defined total order to guarantee correctness. Database replication protocols can also be classified in terms of when the client is notified that the transaction has been committed: in eager (synchronous) replication protocols, the client is only replied to after all replicas have committed the transaction (using, e.g., two-phase commit (2PC)), which can be costlier in terms of latency but provides stronger consistency; lazy (asynchronous) replication protocols reply to the client as soon as the transaction has committed in some replica, later propagating updates to other replicas, providing weaker consistency because of potential temporary divergence between replicas. An alternative definition is to consider whether updates are propagated to other replicas before the transaction is committed at the primary using a primitive that guarantees delivery and the appropriate message order properties needed by the protocol.

Figure 1 depicts distinct strategies for a replicated database system and how replication can be implemented at that level:

- SQL-based, at a middleware layer, above the database engine layer;
- Log shipping, at the database engine layer; and
- Block Device, at the storage layer, below the database engine layer.

Active replication can be implemented above the database engine layer by reliably forwarding SQL statements from clients to all replicas, handling synchronization/recovery at this level, when needed.

HA-JDBC (High Availability JDBC) (Paul Ferraro, 2014) is an open source tool that can be used to implement replication at this level, using the JDBC standard to communicate with the database layer, making it generic. At the database engine layer, passive replication can be implemented by, for example, recording updates in a transactional log that can then be shipped to other replicas and replayed. Conceptually, leader election can be handled either at this layer or above. Active replication can be implemented by sending each operation to other replicas, handling synchronization/recovery at this level. The implementations of replication protocols at this level tend to be tightly coupled to the specific database engine and its internals. At the storage layer, passive replication can be implemented by mirroring block devices between the primary and backup replicas. However, in current solutions such as DRDB (LinBit) (Ellenberg, 2007), while an underlying protocol guarantees detection and recovery of lost updates, there is no notion of transactional context for the blocks, so transaction atomicity/consistency cannot be enforced at this level. Leader election would need to be handled above. Active replication does not make sense at this level.

3 HIGH AVAILABILITY

In the context of the CloudDBAppliance, appliances can be defined as the integration of software with high performance computing hardware to provide a mainframe analogue for data processing. Providing high availability within a data centre means, in this context, providing fast failover between consistent replicas. Minimizing any synchronization overhead during normal operation is a concern, addressed by implementing replication at the middleware level, which makes it independent of the operational database. This decoupling also prevents the design of the operational database from being constrained by replication.

Unlike common replication scenarios, due to the considerable amount of resources that each appliance represents, replicas are limited to two. This precludes approaches based on quorums, which require at least 3 replicas to function correctly. The solution, feasible considering a data centre environment, is to assume the existence of an eventually perfect failure detector (Chandra and Toueg, 1996). In the following sections we describe the proposed replication architecture and protocol.

3.1 Architecture Overview

This section discusses the proposed replication architecture its integration with the operational database. Replication is implemented at the SQL level, as depicted in Figure 2, as a middleware layer, making it transparent to the operational database and client applications. The middleware intercepts SQL statements and is able to perform tasks such as removing non-determinism or imposing a total-order on statements before forwarding the processed statements to the operational database instances as required. This simplifies integration in existing systems, as it requires only a configuration change in JDBC drivers and data sources.

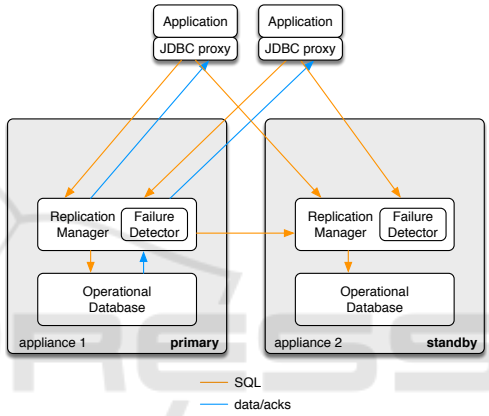

Figure 2: High Availability conceptual architecture.

Figure 2 depicts the proposed architecture for the intra-data centre replication and fault tolerance protocol, considering applications running outside the primary appliance. Appliance components include a Replication Manager (RM), a Failure Detector (FD) and the operational database (opDB). Clients connect to the replication manager through a JDBC proxy. Intercepting requests from client applications to the operational database is key as these contain SQL statements, parameters, and a variety of control commands. Replies return data to the application and report various status information. The JDBC interface provides just that.

3.2 Replication and Fault-Tolerance Algorithms

The replication and fault tolerance protocol is based the state machine depicted in Figure 3, which shows the primary, standby and recovering states, along with the allowed state transitions.

Normal protocol operation is described in Algorithm 1.

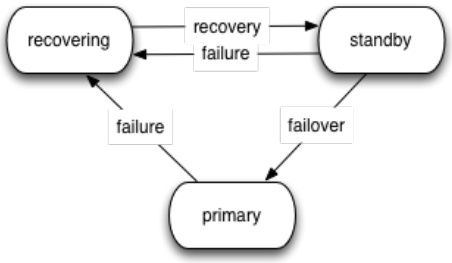

Figure 3: State machine for the replication and fault-tolerance protocol.

Algorithm 1: CDBA replication protocol - regular scenario.

1: Applications send SQL statements to the primary's Replication Manager (RM).
2: Non-determinism in SQL statements is removed by the primary's RM.
3: Statements are logged and reliably sent to the RM of the standby appliance.
4: RMs push SQL statements to their opDB replica.
5: The primary's opDB replies are forwarded to the application via JDBC.

Algorithms 2, 3 and 4 describe how failover and recovery are handled.

Algorithm 2: CDBA replication protocol - failover.

1: The standby's Failure Detector (FD) reliably detects the failure of the primary.
2: Standby takes over, becoming primary.

Regarding Algorithm 3, if the other appliance is offline, the recovering appliance could serve a possibly stale version of the data. Also, the recovering appliance cannot become the primary in this situation to prevent inconsistencies, as the other appliance may have committed transactions before failing itself. Also, a process for breaking ties regarding which becomes the primary, when both appliances become online, is required.

During state transfer, reads can still be served by the primary from a consistent snapshot.

Algorithm 3: CDBA replication protocol - recovery (restart).

1: Appliance starts in the *recovering* state.
2: **if** the other appliance is available **then**
3: **if** this appliance is up-to-date **then** it enters the standby state.
4: **else** a state transfer process ensues.
5: **else** this appliance remains in the *recovering* state.
6: When both appliances become online, one becomes the primary.

Algorithm 4: CDBA replication protocol - recovery (state transfer).

1: The primary pauses transaction processing, queuing requests.
2: The primary sends its state (or missing transactions) to the recovering appliance.
3: Once up-to-date, the recovering replica enters the standby state and normal operation resumes with pending requests.

4 PRELIMINARY RESULTS

In this section, we evaluate a set of JDBC-based mechanisms, considering different configurations, and compare these to a non-replicated baseline. All tests ran over CDBA's operational database (opDB) and none required specific database-level configuration.

Each configuration was tested using the YCSB benchmark with balanced read and write operations (50/50). It should be pointed out that as the number of performed operations increases, the number of records loaded into the database also increases. First, we present an assessment of the overhead each JDBC technology introduces without replication. The selected technologies include an implementation that provides high availability (HA-JDBC) and an implementation that uses a client-server model that matches the architecture proposed in Section 3.

Then we evaluate each mechanism in a replicated setting and compare these with the implementation proposed in Section 3.

4.1 JDBC Overhead

The baseline configuration consists of accessing a single opDB instance directly. HA-JDBC (Paul Ferraro, 2014) enables high-availability by allowing an application to use clusters of identical databases through an embedded client. V-JDBC (Michael Link, 2007), on the other hand, provides a client-server interface for remote databases, offering a selection of transport protocols, but lacks replication awareness. For these experiments, V-JDBC was configured to use Remote Method Invocation (RMI) (Pitt and McNiff, 2001) as the service protocol.

Figure 4 shows throughput for the baseline, HA-JDBC and V-JDBC over a single opDB instance, as the number of operations per second [ops/sec]. Ex-

periments were ran for databases of different sizes: from ten thousand to three million records. Increasing the size of the database seems to have an impact on throughput.

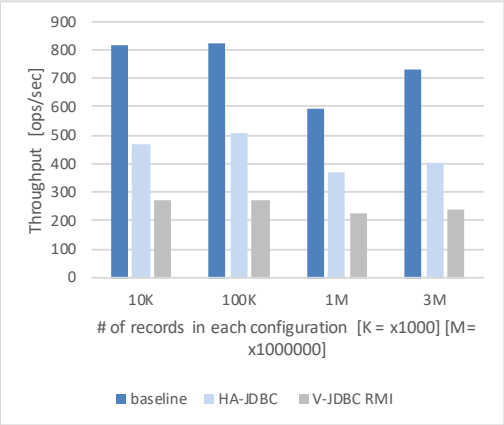

Figure 4: Mean throughput for JDBC proxies under increasing workloads.

Results show that the HA-JDBC middleware, even without replication, imposes a high penalty on throughput, when compared to the baseline (41,15%). V-JDBC imposes an even higher overhead, due to its client/server architecture.

Latency for read operations is depicted in Figure 5, considering the same workloads. Results match those presented for throughput with V-JDBC imposing the highest penalty on latency and still significant overhead introduced by HA-JDBC. Read latency remains stable even as the number of pre-loaded items in the opDB increases.

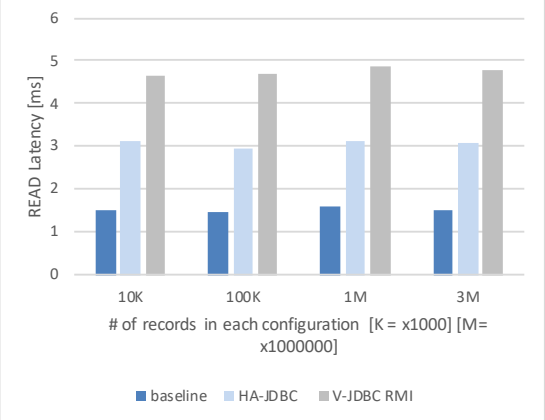

Figure 5: Mean read latency for JDBC proxies under increasing workloads.

Table 1: Evaluated configurations.

Configurations	Description
Baseline	Direct access, 1 opDB instance
HA-JDBC 2	HA-JDBC, 2 opDB instances
V-JDBC HTTP	V-JDBC using HTTP
V-JDBC AUGEO	AUGEO's V-JDBC (HTTP)
CDBA	CDBA with 1 opDB instance
CDBA 2	CDBA with 2 opDB instances

4.2 Evaluation of Replication Mechanisms

The implementation of CloudDBAppliance's Replication Manager merges HA-JDBC with V-JDBC, enabling the system to establish clusters with groups of replicas with the proxy capabilities of V-JDBC. This enables requests to be marshalled and transmitted across a network, delivering them to a remote instance. In this section, we evaluate each component along with the integrated version in non-replicated settings and settings with two replicas. This micro-benchmarking campaign focuses on evaluating replication algorithms during normal operation, i.e., in the absence of faults.

Table 1 provides an overview of the evaluated replication configurations. The baseline is, as in Section 4.1, direct access to a single instance of the CDBA's operational database. We evaluated V-JDBC using HTTP as the transport protocol, including a vendor-specific implementation over HTTP, that uses a different serialization mechanism to marshal packets over the network, Augeo (Augeo Software, 2016). HA-JDBC was evaluated in a replicated configuration, with two opDB instances (HA-JDBC 2). The replication protocol proposed in Section 3, realized in CDBA's Replication Manager was evaluated both in a non-replicated setting and a setting with two opDB replicas (CDBA 1 and CDBA 2, respectively). In either configurations with 2 opDB instances, no additional synchronization mechanisms were configured. This ensures that the requests are replicated solely by the JDBC-based middleware, and does not take advantage of the synchronization mechanisms of the underlying opDB instances. Also, request load-balancing was not enabled, with read requests directed to the primary replica, where applicable. Evaluation of the replication mechanisms was conducted on a scenario with 100K pre-populated records.

Figure 6 shows the throughput achieved by the configurations in Table 1. Comparing results for V-JDBC RMI (Figure 4), V-JDBC HTTP and V-JDBC AUGEO shows a slight improvement for V-JDBC HTTP over V-JDBC RMI, with V-JDBC AUGEO per-

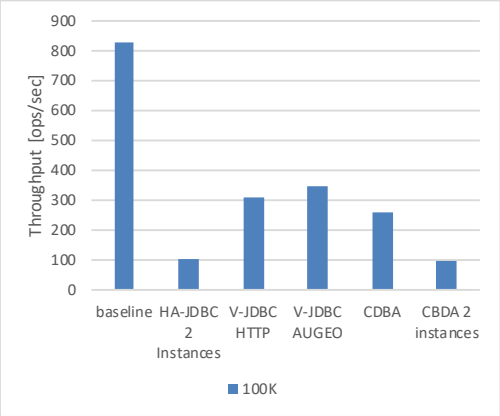

Figure 6: Mean throughput for configurations in Table 1.

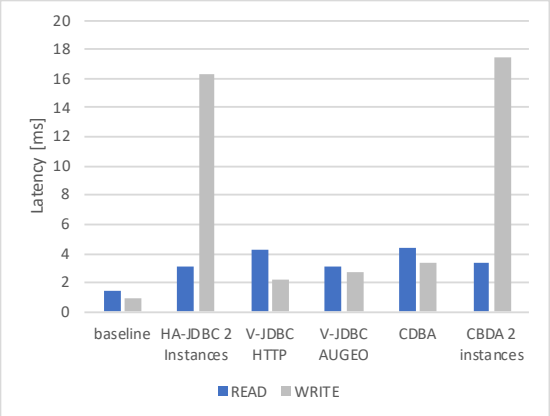

Figure 7: Mean latency of read and write operations for configurations in Table 1.

forming even better. CDBA achieved a throughput of approximately 250 operations per second, lower than that of HA-JDBC and V-JDBC HTTP and V-JDBC AUGEO, but practically on par with V-JDBC RMI. Regarding replicated setups, HA-JDBC 2 and CDBA 2 achieved similar throughput, of approximately 100 operations per second. As CDBA's replication management derives from HA-JDBC this is to be expected. Impact on throughput is justified by the need to maintain two state machine instances: one per opDB instance. Moreover, because an active replication mechanism is being used, to ensure consistency, an answer to the client is only provided once both opDB instances have successfully applied all changes.

Figure 7 shows the mean latency of read and write operations, for the configurations in Table 1.

The impact of marshalling and sending requests across the network is visible both in write and read latencies for V-JDBC HTTP but, to a lesser extent, on V-JDBC AUGEO, when compared to the baseline. This is a consequence of the more efficient serialization mechanism in V-JDBC AUGEO, as, in fact, mean read latency for V-JDBC AUGEO is on par with HA-JDBC 2. Capitalizing on V-JDBC AUGEO's efficiency, mean read latency for the CDBA 2 configuration is also on par with HA-JDBC 2. For more demanding workloads, the ability to use load balancing for read requests will likely yield better read latency in the replicated scenarios, when compared to single-replica scenarios.

The impact of replication is most visible on writes as mean latency for HA-JDBC 2 and CDBA 2 is significantly higher than for single replica configurations. Regarding HA-JDBC 2, several factors contribute to the increase in latency: (1) execution on the standby replica takes place only after the successful execution on the primary; (2) a result is only returned to the client after comparing results from the standby to the primary's. Regarding CDBA 2, overhead is due to: (1) preprocessing the query to remove non-determinism; (2) serialization and network communication (3) ensuring queries are guaranteed to be delivered to both the primary and the standby and in the same order; (4) waiting for both replicas to acknowledge the write was successful before notifying the client.

This benchmarking campaign exposed the overhead imposed by each feature of the replication protocol. First, we showed how using a client-server model for JDBC adds overhead and the impact of different transport protocols, when compared to other JDBC models (HA-JDBC) and the direct access baseline. Second, we showed the performance penalty for introducing replication to the client-server JDBC model. The penalty is most noticeable on write requests, as evidenced by results on write latency. Nevertheless, performance largely follows what is observed for other high-availability providers, notwithstanding CDBA's additional ability to forward JDBC objects across the network. This ability endows CDBA's Replication Manager with extended flexibility in exploring different replication configurations, namely based on multiple communication patterns, as it decouples the client from the operational database instances.

5 CONCLUSION

This paper introduced CloudDBAppliance's replication architecture for high-availability within a data centre, along with a proposed replication protocol. Moreover, it provides a set of micro-benchmarks that evaluate the performance of each component with different configurations, as well as the impact of each

feature. Results show the penalty on throughput and latency of decoupling clients from database instances by leveraging a client-server implementation of JDBC and, particularly for writes, of introducing a replication mechanism for high-availability.

The proposed replication mechanism, that provides high-availability while still decoupling clients from database instances, performs with minimal overhead regarding other proposals for implementing high-availability at the SQL level, namely HA-JDBC.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Horizon 2020 programme – 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

Augeo Software (2016). Augeo v-jdbc: Virtual remote access for jdbc datasources.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267.

Correia Jr, A., Pereira, J., Rodrigues, L., Carvalho, N., Vilaça, R., Oliveira, R., and Guedes, S. (2007). Gorda: An open architecture for database replication. In *Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007)*, pages 287–290. IEEE.

Défago, X., Schiper, A., and Urbán, P. (2004). Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys (CSUR)*, 36(4):372–421.

Ellenberg, L. (2007). Drbd 8.0. x and beyond: Shared-disk semantics on a shared-nothing cluster. *LinuxConf Europe*.

Guerraoui, R. and Schiper, A. (2001). The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41.

Marandi, P. J., Primi, M., and Pedone, F. (2011). High performance state-machine replication. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 454–465. IEEE.

Michael Link (2007). V-jdbc: Virtual remote access for jdbc datasources.

Paul Ferraro (2014). Ha-jdbc: High availability jdbc.

Pitt, E. and McNiff, K. (2001). *Java. rmi: The Remote Method Invocation Guide*. Addison-Wesley Longman Publishing Co., Inc.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319.

Thomson, A. and Abadi, D. J. (2010). The case for determinism in database systems. *Proceedings of the VLDB Endowment*, 3(1-2):70–80.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., and Alonso, G. (2000). Understanding replication in databases and distributed systems. In *Proceedings 20th IEEE International Conference on Distributed Computing Systems*, pages 464–474. IEEE.