International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 11
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 November 2023*

Study on Basics of Replica Control Management in Distributed Data

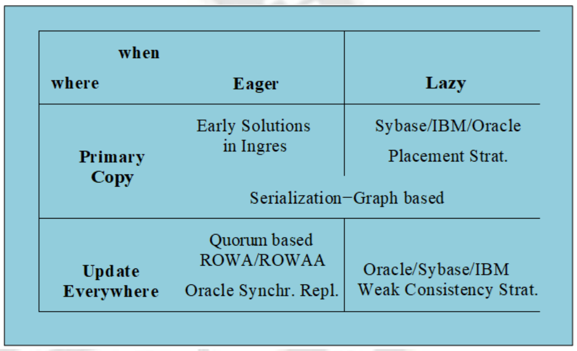
***Manohar Lal Raksha, *Dr. Ajay Jain**
Research Scholar, Dr. APJ Abdul Kalam University-Indore

Abstract- A replicated database system is a distributed system in which each site stores a copy of the database (full replication) or parts of the database (partial replication). Data access is done via transactions. A transaction represents a logical unit of read and write operations. Two important components of a replicated database system are concurrency control and replica control. Concurrency control isolates concurrent transactions with conflicting operations, while replica control coordinates the access to the different copies. This Paper provides an informal introduction to database replication along with an overview of the advantages and disadvantages of traditional solutions. In particular, we elaborate on the main drawbacks of these solutions to be able to distinguish between inherent and avoidable limitations. This leads us to the key concepts behind the approach proposed in this dissertation which eliminate the avoidable and alleviate the inherent limitations of traditional solutions.

Keywords- *Replica, Database System, Read and Write Operations, Concurrency Control, Traditional Approach.*

INTRODUCTION

The strongest correctness criteria for a replicated system is 1-copy-serializability (1CSR) *[BHG87]*: despite the existence of multiple copies, an object appears as one logical copy (1-copy-equivalence) and the execution of concurrent transactions is coordinated so that it is equivalent to a serial execution over the logical copy (serializability). Furthermore, transaction atomicity guarantees that a transaction commits (executes successfully) on all or none of the participating sites despite the possibility of failures. Not all replica control protocols guarantee 1-copy-serializability or atomicity; some provide lower or undefined levels of correctness in order to increase performance. Gray et al. *[GHOS96]* categorize replica control mechanisms according to two parameters: when updates are propagated between the copies, and where updates take place, i.e., which copies can be updated (Table 1). Update propagation can be done within transaction boundaries or after transaction commit. In the first case, replication is eager, otherwise it is lazy. Eager replication allows the detection of conflicts before the transaction commits. This approach provides data consistency in a straight- forward way, but the resulting communication overhead increases response times significantly. To keep response times short, lazy replication delays the propagation of changes until after the end of the transaction, implementing update propagation as a background process. However, since copies are allowed to diverge, inconsistencies might occur. In terms of which copy to update, there are two possibilities: centralizing updates (primary copy) or a distributed approach (update everywhere).



when / where	Eager	Lazy
Primary Copy	Early Solutions in Ingres	Sybase/IBM/Oracle Placement Strat.
	Serialization−Graph based	
Update Everywhere	Quorum based ROWA/ROWAA Oracle Synchr. Repl.	Oracle/Sybase/IBM Weak Consistency Strat.

Table 1: Classification of replica control mechanisms

Communication within the transaction execution time. In lazy schemes the non-trivial problem of reconciliation arises. When two trans- actions update different copies of the same data item and both commit locally before propagating the update, the data becomes inconsistent. Such a conflict must be detected and reconciled.

Traditional Eager Replication- Using eager replication, 1-copy-serializability and atomicity can be achieved in a straightforward way. Replica control is combined with the concurrency control mechanisms, for instance 2-phase-locking (2PL) or timestamp-based algorithms, in order to guarantee serializability. Furthermore, an atomic commitment protocol, like 2-phase-commit (2PC) is run at the end of the transaction to provide atomicity.

Table 1 classifies some of the better-known protocols. Early solutions, e.g., distributed IN- GRES, use synchronous primary copy/site approaches *[AD76, Sto79]*. Most of the algorithms avoid this centralized solution and follow the update everywhere approach guaranteeing 1-copy-equivalence by accessing a sufficient number of copies. A simple approach is read-one/write-all (ROWA) *[BHG87]*, which requires update operations to access all copies while read operations are done locally. This approach has the major drawback of not being fault-tolerant: processing halts whenever a copy is not accessible. To tolerate site failures, read-one/write-all-available (ROWAA) is used, which requires to update only the available copies *[BG84, GSC 83]*. Carey et. al *[CL91]* provide an evaluation of ROWA with different concurrency control mechanisms.

Lazy Replication: Due to the complexity and performance implications of eager replication there exist a wide spectrum of lazy schemes. Naturally, lazy replication reduces response times since transactions can be executed and committed locally and only then updates are propagated to the other sites. However, 1-copy-serializability can only be guaranteed in very restricted primary copy configurations. Some lazy schemes only ensure that all replicas of a data item eventually converge to a single final value and do not consider that transactions create dependencies between the values of different data items. Atomicity cannot be guaranteed at all. If a node fails before it propagates the updates of a committed transaction to the other sites, then is lost. Many lazy schemes use a primary copy approach. This means that update transactions must be submitted at the site with the corresponding primary copies and transactions which want to update data items whose primary copies reside at different sites, are not allowed.

Replication in Commercial Database: Clearly, commercial databases favor lazy propagation models (see Table 1). Most systems started with a primary copy approach specialized for either OLTP (On Line Transaction Processing) or OLAP (On Line Analytical Processing) *[Sta94, Gol94]*. In the meanwhile, many of the big database vendors provide a whole spectrum of primary copy and update everywhere approaches.

Sybase Replication Server provides an extended publish-and-subscribe scheme and clearly favors a primary copy approach although update everywhere configurations are possible. Up-dates are propagated to the other copies immediately after the commit of the transaction. The updates are obtained from the log as soon as the log records are stored on disk. This push strategy is an effort to minimize the time that the copies are inconsistent and an implicit acknowledgment of the importance of keeping copies consistent in an OLTP environment. In the primary copy configuration, updates can either be done by synchronously connecting to the primary site or asynchronously by transferring procedure calls between the site that wants to update the item and the primary site. In their update everywhere configuration, updates may take place at any site and conflict resolution has to be done by the application. IBM Data Propagator was first a primary copy approach geared towards OLAP and mobile architectures. It adopted a pull strategy in which updates were propagated only at the client request, which implies that a client will not see its own updates unless it requests them from the central copy. Having OLAP applications in mind, those requests may range from simple point-in-time refreshes and continuous update notifications to sophisticated subscriptions for aggregate data. Over the years, IBM enhanced the system to also support update everywhere providing conflict detection and automatic compensation. IBM also uses the log information to detect updates, and, to optimize the process, can even capture log records directly from the memory of the database system.

Lazy Replication with Lower Levels of Consistency: From the research point of view, there has also been considerable work in lazy replication. Early papers provide the user with a way to control inconsistency, i.e., although the data may be obsolete or even inconsistent, the degree to which the data may be “wrong” is limited and well- defined. A couple of weak consistency models have been constructed that provide correctness criteria weaker than 1-copy-serializability. Examples of weak-consistency replication models are Epsilon-serializability *[PL91]* and N-Ignorance *[KB91]*. Epsilon-serializability measures the distance between database objects like the difference in value or the number of updates applied. The application can therefore specify the amount of inconsistency tolerated by a transaction. N-Ignorance is based on quorums. It relaxes the requirement that quorums must intersect in such a way that the inconsistencies introduced by concurrent transactions are bounded. The replication system in Mariposa *[SAS 96]* builds an economic framework for data replication. The frequency of update propagation depends on how much the maintainer of a replica is willing to pay. Also the staleness of the data in a query is determined by the price a user wants to pay. For all these approaches, however, making the choice of the right bound of inconsistency is a non-trivial problem and users must have a good understanding of the inconsistency metrics.

Lazy Replication providing 1-Copy-Serializability: More recent work has explored the possibility of using lazy replication while still providing 1- copy-serializability. Thus, [CRR96] have shown that even in lazy primary copy schemes, serializability cannot be guaranteed in every case. The way to get around this problem is to restrict the placement of primary and secondary copies across the system. The main idea is to define the set of allowed configurations using configuration graphs where nodes are the sites and there is a non-directed edge between two sites if one has the primary copy and the other a secondary copy for a given data item. If this graph is acyclic serializability can be guar- anteed by simply propagating updates sometime after transaction commit *[CRR96]*. Pacitti et al. *[PSM98, PMS99]* have enhanced these initial results by allowing certain cyclic configurations. These configurations, however, require more complex update propagation schemes, namely, up- dates to secondary copies must be executed in causal or the same total order at all sites. Breitbart et al. *[BKR 99]* propose an alternative solution by requiring the directed configuration graph (edges are directed from primary copy to secondary copy) to have no cycles. This also requires to intro- duce more sophisticated update propagation strategies.

Combining Eager and Lazy Replication: A further primary copy approach combining eager and lazy techniques has been proposed in *[BK97, ABKW98].* The system is eager since the serialization order is determined before the commit of the transactions (using distributed locking or a global serialization graph). This means that communication takes place within the execution time of each transaction. However, the system can be called lazy because within the boundaries of the transaction the execution of the operations only takes place at one site. Propagating the updates to the remote copies is only after the commit and there is no 2-phase commit.

Replication in Non-Database Systems: There exist many lazy replication solutions that have not evolved with the concept of transactions in mind but in a more general distributed setting, for instance, distributed file systems, replication on the web *[RGK96]*, document replication *[ARM97],* and so forth. A good survey of early approaches can be found in *[CP92].* In these environments, lazy replication provides more easily the requested level of consistency because transactional dependencies do not need to be considered.

Replication and Distributed Computing: In distributed computing, the workload is distributed among several off-the-shelf workstations connected by a fast network. Distributed computing is used for scalability and fault-tolerance. Whenever the workload increases more nodes are added to the system in order to increase the process capacity. Furthermore, distributed computing provides Contention management if the failure of one site does not hinder the execution at the other sites. It might even be possible for the available nodes to take over the work of failed nodes.

Problems of Traditional Eager Replication and how to avoid them: Although eager update everywhere replication is the adequate choice from a theoretical point of view, current solutions are not attractive options in terms of performance and complexity. The question to ask is whether their limitations are completely inherent to the eager, update everywhere model or whether some of them are only an artifact of the mechanisms used. In what follows, we discuss some of the typical mechanisms found in traditional approaches, how they influence performance and complexity, and how their drawbacks can be circumvented by applying adequate techniques.

CONCLUSIONS

As it is done with different levels of isolation, we propose to weaken full correctness to provide faster solutions. We allow a local site to commit a transaction whenever the global serialization order has been determined and do not require that it waits for the other sites to execute the trans- action. Instead, the local site relies on the fact that the other sites will serialize the transaction in the same way according to the total order in which write sets are delivered. Furthermore, we exploit the different degrees of reliable message delivery provided by group communication systems in order to determine the overall correctness in failure cases.

REFERENCES:

[1] **[AD76]** P. A. Alsberg and J. D. Day. A principle for resilient sharing of distributed re- sources. In Proc. of the Int. Conf. on Software Engineering, pages 562–570, San Francisco, California, October 1976.

[2] **[ARM97]** G. Alonso, B. Reinwald, and C. Mohan. Distributed data management in workflow environments. In Proc. of the Int. Workshop on Research Issues in Data Engineer- ing (RIDE), Birmingham, United Kingdom, April 1997.

[3] **[BHG87]** P. A. Bernstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Database Systems. Addison Wesley, Massachusetts, 1987.

[4] **[BG84]** P.A. Bernstein and N. Goodman. An algorithm for concurrency control and recovery in replicated distributed databases. ACM Transactions on Database Systems, 9(4):596–615, December 1984.

[5] **[BK97]** Y. Breitbart and H. F. Korth. Replication and consistency: Being lazy helps some- times. In Proc. of the ACM Symp. on Principles of Database Systems (PODS), pages 173–184, Tucson, Arizona, May 1997.

[6] **[GHOS96]** J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The dangers of replication and a solution. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 173–182, Montreal, Canada, June 1996.

[7] **[CL91]** M. J. Carey and M. Livny. Conflict detection tradeoffs for replicated data. ACM Transactions on Database Systems, 16(4):703–746, 1991.

[8] **[CRR96]** P. Chundi, D. J. Rosenkrantz, and S. S. Ravi. Deferred updates and data placement in distributed databases. In Proc. of the Int. Conf. on Data Engineering

(ICDE), pages 469–476, New Orleans, Louisiana, February 1996.

[9] **[PL91]** C. Pu and A. Leff. Replica control in distributed systems: An asynchronous approach. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 377–386, Denver, Colorado, May 1991.

[10] **[KB91]**]N. Krishnakumar and A. J. Bernstein. Bounded ignorance in replicated systems. In Proc. of the ACM Symp. on Principles of Database Systems (PODS), pages 63–74, Denver, Colorado, May 1991.

[11] **[SAS96]** J. Sidell, P. M. Aoki, A. Sah, C. Staelin, M. Stonebraker, and A. Yu. Data replication in Mariposa. In Proc. of the Int. Conf. on Data Engineering (ICDE), pages 485– 494, New Orleans, Louisiana, February 1996.

[12] **[Sta94]** D. Stacey. Replication: DB2, Oracle, or Sybase. Database Programming & Design, 7(12), 1994.

[13] **[PSM98]** E. Pacitti, E. Simon, and R. N. Melo. Improving data freshness in lazy master schemes. In Proc. of the Int. Conf. on Distributed Computing Systems (ICDCS), pages 164–171, Amsterdam, The Netherlands, May 1998.

[14] **[PMS99]** E. Pacitti, P. Minet, and E. Simon. Fast algorithms for maintaining replica consis- tency in lazy master replicated databases. In Proc. of the Int. Conf. on Very Large Data Bases (VLDB), pages 126–137, Edinburgh, Scotland, September 1999.

[15] **[RGK96]** M. Rabinovich, N. H. Gehani, and A. Kononov. Scalable update propagation in epidemic replicated databases. In Proc. of the Int. Conf. on Extending Database Technology (EDBT), pages 207–222, Avignon, France, March 1996.