Quorum Systems in Replicated Databases: Science or Fiction?

Avishai Wool
Bell Labs, Lucent Technologies,
Murray Hill, New Jersey, USA
E-mail: `yash@acm.org`
`http://www.bell-labs.com/~yash/`

Abstract

A quorum system is a collection of subsets of servers, every two of which intersect. Quorum systems have been suggested as a tool for concurrency control in replicated databases almost twenty years ago. They promised to guarantee strict consistency and to provide high availability and fault-tolerance in the face of server crashes and network partitions. Despite these promises, current commercial replicated databases typically do not use quorum systems. Instead they use mechanisms which guarantee much weaker consistency, if any. Moreover, the interest in quorum systems seems to be waning even in the database research community.

This paper attempts to explain why quorum systems have not fulfilled their old promises, and at the same time to argue why the current state of affairs may change. As technological advances bring new capabilities, and new applications bring new requirements, the time may have come to review the validity of some long standing criticisms of quorum systems.

Another theme of this paper is to argue that if quorum systems are to play a role in database research, it is not likely to be for their claimed fault-tolerance capabilities. Rather, more attention should be given to a somewhat overlooked feature of quorum systems: they allow load balancing among servers while maintaining strict consistency. Thus quorum systems may offer a way to scale up the throughput of heavily loaded servers.

1 Introduction

A *quorum system* is a collection of subsets (quorums) of servers, every two of which intersect. During the late 70's and early 80's, quorum systems were proposed as a basic mechanism for concurrency control in replicated databases [Tho79, Gif79, GB85, Mae85, Her86]. Informally, a quorum-based replication scheme works as follows. Data items are timestamped and written to some quorum of servers. In order to read a data item, the reader accesses the copies held by a (possibly different) quorum of servers, and uses the value with the highest timestamp. The quorum intersection property guarantees that at least one server observes both operations, so the reader

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

in fact obtains the most up-to-date value. It is easy to see that this approach, coupled with standard two-phase protocols, ensures that strict consistency can be maintained and that the transactions can be globally serialized.

Quorum systems were attractive to database researchers mainly because they offer a de-centralized approach that tolerates failures. For instance, quorum-based databases are able to maintain their consistency even in the presence of network partitions [DGS85] and server crashes (as quantified by their fault-tolerance [BG86]). However, the quorum approach's promise of high availability was arguably its most desirable feature, and it certainly generated a substantial amount of research (e.g., [BG87, ET89, PW95, AW98]). Availability is the probability that the member servers of at least one quorum are functioning, assuming that the servers crash independently with a fixed probability, and that the complete database is replicated at every server. Many quorum systems, such as those of [KC91, Kum91, AE91, RST92, PW97b, PW97a, Baz96], exhibit very high availability, and moreover, their availability tends to 1 very rapidly as the system scales up and more servers are added.

Despite these features, quorum systems are not in wide-spread use within commercial replicated databases. As an example, we can consider the Oracle8 product [Ora97], which provides several modes of data replication. Among these, the only mode that guarantees serializability, called Synchronized Data Propagation, uses a Read-One-Write-All (ROWA) approach (cf. [HHB96]). Oracle's product documentation in fact recommends *not* to use this mode because "... it can function only when all sites in the system are concurrently available" [Ora97, p. 30–26], i.e., it has very poor write-availability. All the other modes of replication supported by Oracle8 essentially take a lazy replication, primary-copy approach, coupled with various heuristics which attempt to detect and resolve some common conflict scenarios. It is well known that such schemes do not guarantee serializability, and in fact serializability can only be guaranteed if the data placement graph is acyclic [CRR96].

There are some valid reasons why quorum systems have not yet been used in replicated databases. In the next section we shall list the more common criticisms, and discuss whether they are still valid today. Then in Section 3 we contrast the situation in the database world with other domains where quorum systems have been successfully deployed. In Section 4 we suggest that by shifting the focus from fault-tolerance to improved performance, we can realize that the load-balancing capability of quorum systems has significant, yet overlooked, potential. We also briefly touch upon some new applications that may benefit from a quorum-based replicated database, and we conclude with Section 5.

2 Common Arguments Against Quorum Systems

2.1 The Local Read Paradigm

One of the most common arguments against quorum systems is that they require accessing one or more remote copies for read operations, and that incurring a network delay for reads is prohibitively expensive. If this view is adopted, we reach the *local read paradigm:* read operations are only allowed to access items that are stored on the local machine. An immediate consequence is that variants of ROWA become the only allowable replication schemes, and in particular quorum-based replication is excluded.

However, technological advances are working against this argument, since data networks are improving faster than disks are. For instance, the bandwidth available on local area networks (LANs) has improved from around 10Mbps to around 1Gbps—a 100 fold increase—over the last decade, whereas disk transfer rates only improved from about 1Mbps to 10Mbps over the same period. As for latency, network latency is mostly due to protocol software which improves with CPU speed, whereas disk latency is limited by the mechanics of the seek time and the disk rotation speed [GR93, pp. 53,59].

In the setting of LANs there should be little doubt that the local read paradigm is no longer justified. In fact, a "local" read often includes a LAN access anyway, since the client software usually runs on a separate machine rather than directly on the DB server machine. Moreover, LANs typically have hardware-based broadcast communication primitives, so sending a read request (such as a high level SQL command) to a quorum of servers can be achieved by a single message. The multiple reads all execute in parallel, thus the reader would not even suffer a substantially longer delay. To some extent, remote reads over a LAN may be reasonable even for main memory databases [GS92, JLR+94], or if the data is held in the cache; the incurred networking delay would be more than that of a pure memory access, but still much less than a disk access.

Current wide area networks (WANs) are still somewhat slower than LANs (e.g., 155Mbps for an OC3 line), and the latency in a WAN depends on the geographical distance and on the number of routers between the end-points. Nevertheless the argument that networks improve faster than disks still holds. This is especially true in a private network or a virtual private network (VPN), where quality of service can be guaranteed. So the validity of the local read paradigm is becoming more questionable in the wide-area setting too.

There is some evidence that at least the database research community may be moving away from the local read paradigm. Over the last two years we have seen the emergence of several replication schemes in the database literature, schemes which do not adhere to the local read paradigm quite so strictly. For instance, the protocol of [GHOS96] includes obtaining locks from remote machines even for read-only transactions, and the protocols of [BK97] include obtaining read, write, and commit permissions from a replication-graph-manager machine. These protocols have yet to be implemented in an actual database, but simulations of their performance over WANs [ABKW98] show that the network is not a bottleneck for either protocol.

We can conclude that while the local read paradigm was a valid objection to quorum-based replication for many years, its validity is diminishing with time. Remote reads should no longer be considered to be prohibitive *a priori*. In many settings they are quite reasonable.

2.2 Reads vs. Writes

Another criticism of quorum systems is that when read operations are much more frequent than write operations, which is a typical scenario, optimizing the read operations is more important than optimizing the writes [GR93]. In such scenarios the total amount of work incurred by a ROWA approach is claimed to be lower than that of a quorum-based approach (where total work is measured by the total number of disk accesses or total number of messages per operation).

Since this is a quantitative issue, we can perform the following back-of-the-envelope calculation. Assume a system of n machines, in which writes comprise an f_w fraction of the submitted operations and reads comprise the other $1 - f_w$ fraction. Then on average a quorum-based approach with read quorums of r machines and write quorums of w machines would incur a total work of

$$E_{Quorum} = (1 - f_w) \times r + f_w \times w \tag{1}$$

per operation. In comparison, a ROWA approach, whether lazy or not, would incur a total work of

$$E_{ROWA} = (1 - f_w) \times 1 + f_w \times n. \tag{2}$$

Simple manipulations yield that $E_{Quorum} < E_{ROWA}$ when

$$f_w > \frac{r-1}{n-w+r-1}, \tag{3}$$

and if we assume further that read and write quorums have equal size we end with the condition

$$f_w > \frac{r-1}{n-1}. \tag{4}$$

So we see that one approach is not universally better than the other. The choice depends not only on the frequency of write operations, but also on the ratio of read quorum size to number of replicas.

For instance, using Maekawa's quorum system [Mae85], if $n = 7$ and $r = w = 3$, a quorum-based approach incurs less work than ROWA when 33% of the operations are writes. If $n = 13$ and $r = w = 4$ then the crossover point is 25% writes. In general, $r = w \approx \sqrt{n}$ for this system, so the frequency of writes needs only to exceed $1/\sqrt{n}$ for the quorum-based approach to be advantageous. Quorum systems with variable-sized quorums, such as those of [AE91, PW97a], offer the possibility of outperforming ROWA for even lower write rates, by directing reads to smaller sized quorums, which can be as small as $r \approx \log n$ for both systems.

Clearly, as the number of replicas grows, quorum systems become advantageous for lower write rates. But even for reasonably small systems and moderate write rates, formula (4) shows that quorum-based systems may outperform ROWA-based ones according to the total work measure.

2.3 A Reality Check on Availability

The promise of high availability has been an important part in the development of the theory of quorum systems. However in practice this promise seems not to have materialized as predicted by the theory. The reason is that several seemingly benign modeling assumptions are made in order to make the analysis of availability tractable. Unfortunately, these assumptions clash with the reality of network environments in crucial ways. The standard probabilistic assumptions which underly the definition of availability, as analyzed in [BG87, PW95] and others, are:

1. Crashes are independent events.
2. The communication network is fully connected and never fails.
3. All the servers have the same crash probability p.

Under these assumptions, the availability of a quorum system $\mathcal{Q}$ is the probability of the event that there exists some quorum $Q \in \mathcal{Q}$ for which all the servers $s \in Q$ are functioning.

Note that assumption 2 does not in itself conflict with the fact that quorum systems guarantee strict consistency despite partitions [DGS85]. One may hope that assuming perfect communication, for the purpose of availability analysis, would not change the qualitative predictions of the model. As we shall see, though, this modeling is problematic.

Assumption 3 (uniform crash probability) simplifies the analysis but is not really crucial. It was shown by [SB94, AW98] that essentially the same properties hold with or without it, namely that the optimal availability quorum system is defined by weighted voting, where the optimal weights depend on the individual crash probabilities. If all the servers have the same crash probability p this voting system reduces to either a simple majority voting, which has optimal availability when $p < 1/2$ [BG87], or to a monarchy (single server) when $p > 1/2$ [PW95].

Assumptions 1 and 2 are not so easily dealt with, as seen from the experiments of [AW96]. These experiments measured crashes and network connectivity over a period of 6 months, on a system that consisted of 14 machines on 3 LANs, split between two geographical locations 50 km apart, using Internet (UDP) communications. The experiments measured application-level "crashes": Whenever the measuring daemon on some machine could not function, for whatever reason, a crash was counted.

These experiments show that within a LAN, it is quite reasonable to assume that the communication is perfect—but the crash independence assumption is misleading. Crashes of machines on a LAN were found to be *positively correlated*, which is explained by a dependence on common file systems, domain name servers (DNS), administrative staff, power supply, and so forth. Such a positive crash correlation has a dramatic effect on the availability: In the extreme case, if all the machines crash and revive at identical times, any quorum system will have precisely the same availability as that of a single server. In fact [AW96] observe that the most highly available machine on one of the LANs was down 1.25% of the time, yet using a majority-based quorum system with this machine and 5 others on the same LAN only reduced the unavailability to 1.12%. This is a far smaller boost than that predicted under the independent crashes assumption.

As for WANs, the reverse situation occurs. The experiments of [AW96] show that crash independence between geographically distant sites is a valid assumption—but that the perfect communication assumption is not. The experimental system was partitioned into two or more disconnected components during 6.5% of the time. Thus the portion of the Internet that took part in the study is better modeled by a tree of fully-connected clusters, where each cluster corresponds to a LAN. In such a non-fully-connected network model, it is known that the optimal-availability quorum system is completely contained in a single bi-connected component [INK95]. And indeed the experiments of [AW96] showed that confining the quorum system to 6 machines on a single LAN gave *better* availability than using all 14 machines and allowing quorums to span the two sites.

Thus the shortcomings of the crash independence and perfect communication assumptions form a double-edged sword, which severely limits the predictive power of the standard availability model. On one hand, if all the replicas are placed on a single LAN, crash correlations cancel most of the availability boosting that quorum systems claim to have. On the other hand, if the replicas are placed in geographically distant sites, network partitions have an even more damaging effect.

There are several ways to avoid these problems, but they are typically unpleasant. Crash correlation on a LAN can be decreased by duplicating and separating all the common resources, such as file systems, DNS, power supply, etc., which is costly and hard to manage. Network partitions are very hard to control over the public Internet, since currently nothing can be guaranteed in terms of quality of service. Therefore a high-availability wide area system would probably need to operate its own private high-connectivity network—again an expensive prospect. The latter approach is the one taken by some successful systems (see Section 3), but it is hardly within every organization's budget.

The conclusion we can draw here is somewhat sobering. If quorum systems are to play a significant role in the future of replicated databases, it will not be because they offer high availability, but for some other reason. In Section 4 we shall see one possible such reason, which is the capability of quorum systems to support extremely high load, in a scalable way, without weakening the database's consistency.

3 Some Success Stories

3.1 VAX Clusters and Financial Applications

Quorum systems are used deep within the file system of Digital's OpenVMS operating system, as a basic component of the VAX Cluster technology [VMS]. A VAX Cluster of machines will function as long as a quorum (majority) of the machines are alive and able to communicate with each other. If the live machines in the cluster do not constitute a connected quorum, they will freeze all activity and wait for a quorum to be re-established. This assures that a cluster that has been partitioned will not allow access to the same data from multiple, disconnected machines, since at most one component of a partition (one that still has a quorum) can write to the data. This guarantees proper access to files across the entire distributed file system.

The communication networks that are typically used in VAX Cluster systems are either LANs, or FDDI-based metropolitan-wide networks, running the DECnet communication protocols. Fairly large cluster configurations, comprising dozens of servers (the limit is 96 machines), are in actual use.

This technology is especially attractive to financial applications, for the following reasons: (i) strict data consistency is crucial; (ii) high reliability is required against disasters such as the 1993 World Trade Center explosion and the fire at the Credit Lyonnais bank [CL98]; and (iii) large financial institutions are able to bear the cost of the dedicated communication networks.

Note that the VAX Cluster's quorum system does not provide a quorum-based replicated database *per se*. The system supports file-system semantics but does not provide transaction processing. And in fact the rdb database product, which is now owned by Oracle but originated on Digital's VMS systems [RDB98], uses the VAX Cluster's lock manager but does not use a quorum-based replication scheme.

3.2 Group Communication Systems

Another successful application of quorum systems is in the context of reliable group communication services such as Isis [BvR94], Transis [ADKM92], Totem [AMM+95], Horus [vRBM96], and others. These are fault-tolerant communication middleware subsystems, that provide guarantees on the *order* of messages that are delivered to servers in a group. For instance, such systems are able to guarantee that all the servers in a group receive all the messages sent to the group in exactly the same order (the so called "total order" guarantee). Group communication services typically run over TCP/IP or UDP, both in local and wide area settings.

In order to tolerate network partitions while maintaining the ordering guarantees, such systems need to be able to detect partitions. Once a partition is detected, messages continue to be delivered as usual only in one network component, called the primary component, and other components are essentially blocked. The primary component is defined in terms of a quorum system: at most one network component can contain a complete quorum of servers.

Group communication systems have made their way from the laboratories of academia where they were invented into the commercial world. Critical, real life applications such as the French air traffic control systems and the Swiss stock exchange now use them [Bir98].

Nevertheless, these systems do not provide transaction processing semantics. Typically they do not even make their internal quorum system structure visible through their application interface (API), e.g., they do not provide a "multicast to a quorum" service. However it seems possible to architect a quorum-based replicated database using reliable group communication as a component.

4 Quorum Systems After All?

4.1 Load Balancing Rather Than Availability

One feature of quorum systems, which has attracted some recent theoretical interest, deals with the notion of *load* [NW98]. The crucial observation is that clients can read or write from *any* quorum, and there is a great deal of freedom in choosing which access *strategy* to use. If the quorum selection strategy is properly randomized, then each server only needs to handle a small fraction of the incoming operations, yet strict consistency is still guaranteed.

Specifically, [NW98] shows that for many quorum systems the load on each server can be made as low as $1/\sqrt{n}$ of the operations in an n-server system. This implies that as the system scales up and more servers are added, the load on each one of them actually decreases. This is in stark contrast to ROWA schemes, where the load caused by reads can be balanced, but every server sees 100% of the write operations; and adding servers favors reads even more, but makes matters worse for writes by increasing the write overhead.

The load balancing consideration brings us to the proposal to use quorum systems as performance-enhancing mechanisms, rather than as availability-enhancing ones. One can envision a quorum-based replicated database, that consists of many relatively weak servers connected by a high speed LAN or switch, that would provide: (i) strict consistency (transactional serializability), (ii) shared data item ownership, write anywhere capabilities, and (iii) and extremely high transaction rates.

This architecture may or may not be more highly available than a single server (depending on how independent the failures of the servers can be). However the architecture would be highly scalable, and possibly cheaper to build than a single monolithic high-end server. Applications that may benefit from such an architecture are those that have an extremely high transaction rate, and a substantial fraction of writes (say 10–15% for a 50–60 server system, according to formula (4)).

4.2 New Applications for Quorum Systems

A different motivation for building quorum-based replicated databases comes from new, large scale, distributed applications that have intrinsic survivability and security requirements. Such applications include, for instance, public-key infrastructures (PKI), and national election systems (see [MR98b] for more details). These applications need to be able to tolerate malicious attacks on the servers; For instance, a vote-counting client needs to be able to ignore bogus votes that are reported by a compromised voting server.

A natural modeling of malicious servers is that of Byzantine-faulty processes: In a Byzantine failure model, a bounded number of servers is allowed to send arbitrary messages and not to adhere to the required protocol, yet the protocol needs to remain correct nonetheless (cf. [Lyn96]). Note that in a Byzantine environment, no single server can be trusted, so both the local read paradigm and the ROWA approach are unacceptable.

Quorum systems have recently been adapted to mask Byzantine failures in [MR98a], and studied further in [MRW97]. These so called Byzantine quorum systems provide a mechanism for meeting security requirements, since they allow clients to mask out malicious or erroneous responses from servers, while still enjoying all the properties of "normal" quorum systems, such as strict consistency and excellent load-balancing capabilities.

The Phalanx system of [MR98b] is a software system that uses Byzantine quorum systems to address the above-mentioned requirements. In its current state it only provides the semantics of shared variables, without transactional serializability. However it seems possible, and useful, to extend this approach to full transactional semantics.

5 Conclusions

Quorum systems have been proposed as a tool for concurrency control in replicated databases almost twenty years ago. Our understanding of quorum system theory has developed into a substantial body of work, but they were not implemented in actual databases for a variety of valid reasons.

We have seen that technological advances have undermined the validity of some of these reasons. We have also seen that changing the focus from fault-tolerance to improved performance suggests that quorum systems may have yet-untapped potential for certain types of applications, and we speculated about a possible architecture that can capitalize on the load-balancing capability of quorum systems. Finally, we have seen the emergence of new applications, that benefit from a quorum-based approach.

Thus we can conclude that, at the very least, further research is needed, to test whether the potential of quorum systems can be brought to fruition in real systems. In particular it would be very interesting to see if transaction processing can be combined with ideas and techniques from group communication services to build a quorum-based replicated database.

Acknowledgments

I have had stimulating discussions about quorum systems, and their use in replicated databases (or lack thereof), with many people. In particular I am indebted to Yuri Breitbart, Amr El-Abbadi, Hector Garcia-Molina, Hank Korth, Dennis Shasha, Avi Silberschatz, and Michael Rabinovich, for sharing their views with me. I thank Alex Linetski for giving me a brief under-the-hood introduction to VAX Clusters, and I am grateful to Dahlia Malkhi for her comments on an early draft of this paper.

References

[ABKW98] T. Anderson, Y. Breitbart, H. F. Korth, and A. Wool. Replication, consistency, and practicality: Are these mutually exclusive? In *Proc. ACM SIGMOD Inter. Conf. Management of Data*, pages 484–495, Seattle, June 1998.

[ADKM92] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication subsystem for high availability. In *Proc. 22nd IEEE Symp. Fault-Tolerant Computing (FTCS)*, pages 76–84, 1992.

[AE91] D. Agrawal and A. El-Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Trans. Comp. Sys.*, 9(1):1–20, 1991.

[AMM+95] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. *ACM Trans. Comp. Sys.*, 13(4), 1995.

[AW96] Y. Amir and A. Wool. Evaluating quorum systems over the Internet. In *Proc. 26'th IEEE Symp. Fault-Tolerant Computing (FTCS)*, pages 26–35, Sendai, Japan, 1996.

[AW98] Y. Amir and A. Wool. Optimal availability quorum systems: Theory and practice. *Inform. Process. Lett.*, 65:223–228, 1998.

[Baz96] R. A. Bazzi. Planar quorums. In *Proc. 10'th Inter. Workshop on Dist. Algorithms (WDAG)*, Bologna, Italy, October 1996.

[BG86] D. Barbara and H. Garcia-Molina. The vulnerability of vote assignments. *ACM Trans. Comp. Sys.*, 4(3):187–213, 1986.

[BG87] D. Barbara and H. Garcia-Molina. The reliability of vote mechanisms. *IEEE Trans. Comput.*, C-36:1197–1208, October 1987.

[Bir98] K. P. Birman. Talk at Bell Labs, Murray Hill, NJ, October 1998.

[BK97] Y. Breitbart and H. F. Korth. Replication and consistency: Being lazy helps sometimes. In *Proc. 16th ACM SIGACT-SIGMOD Symp. Princip. of Database Systems (PODS)*, pages 173–184, Tucson, Arizona, May 1997.

[BvR94] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[CL98] Credit Lyonnais: VMS clusters' trial by fire, April 1998. Available from `http://www.success-stories.digital.com/css/cgi-bin/cssextusr/s=sublist?Customer+Name=Credit+Lyonnais`.

[CRR96] P. Chundi, D. J. Rosenkrantz, and S. S. Ravi. Deferred updates and data placement in distributed databases. In *Proc. 12th IEEE Int. Conf. Data Engineering*, New Orleans, Louisiana, 1996.

[DGS85] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, 1985.

[ET89] A. El-Abbadi and S. Toueg. Maintaining availability in partitioned replicated databases. *ACM Trans. Database Sys.*, 14(2):264–290, June 1989.

[GB85] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *J. ACM*, 32(4):841–860, 1985.

[GHOS96] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. ACM SIGMOD Inter. Conf. Management of Data*, pages 173–182, Montreal, Quebec, 1996.

[Gif79] D. K. Gifford. Weighted voting for replicated data. In *Proc. 7th Symp. Oper. Sys. Princip.*, pages 150–159, 1979.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA, 1993.

[GS92] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *IEEE Trans. Knowledge and Data Eng.*, 4(6):509–516, December 1992.

[Her86] M. Herlihy. A quorum-consensus replication method for abstract data types. *ACM Trans. Comp. Sys.*, 4(1):32–53, February 1986.

[HHB96] A. A. Helal, A. A. Heddaya, and B. B. Bhargava. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, 1996.

[INK95] T. Ibaraki, H. Nagamochi, and T. Kameda. Optimal coteries for rings and related networks. *Distributed Computing*, 8:191–201, 1995.

[JLR+94] H. V. Jagadish, D. Lieuwen, R. Rastogi, A. Silberschatz, and S. Sudarshan. Dali: A high performance main-memory storage manager. In *Proc. 20th Inter. Conf. on Very Large Databases (VLDB)*, 1994.

[KC91] A. Kumar and S. Y. Cheung. A high availability $\sqrt{n}$ hierarchical grid algorithm for replicated data. *Inform. Process. Lett.*, 40:311–316, 1991.

[Kum91] A. Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Trans. Comput.*, 40(9):996–1004, 1991.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[Mae85] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comp. Sys.*, 3(2):145–159, 1985.

[MR98a] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

[MR98b] D. Malkhi and M. Reiter. Secure and scalable replication in Phalanx. In *Proc. 17th IEEE Symp. on Reliable Distributed Systems*, pages 51–58, October 1998.

[MRW97] D. Malkhi, M. Reiter, and A. Wool. The load and availability of Byzantine quorum systems. In *Proc. 16th ACM Symp. Princip. of Distributed Computing (PODC)*, pages 249–257, August 1997.

[NW98] M. Naor and A. Wool. The load, capacity and availability of quorum systems. *SIAM J. Computing*, 27(2):423–447, April 1998.

[Ora97] *Oracle8 Server Concepts*, release 8.0, chapter 30: Database replication, June 1997. Available from `http://www.oracle.com/support/documentation/oracle8/SCN80.pdf`.

[PW95] D. Peleg and A. Wool. The availability of quorum systems. *Information and Computation*, 123(2):210–223, 1995.

[PW97a] D. Peleg and A. Wool. The availability of crumbling wall quorum systems. *Discrete Applied Math.*, 74(1):69–83, April 1997.

[PW97b] D. Peleg and A. Wool. Crumbling walls: A class of practical and efficient quorum systems. *Distributed Computing*, 10(2):87–98, 1997.

[RDB98] Oracle rdb information, 1998. Available from `http://www.oracle.com/products/servers/rdb/index.html`.

[RST92] S. Rangarajan, S. Setia, and S. K. Tripathi. A fault-tolerant algorithm for replicated data management. In *Proc. 8th IEEE Int. Conf. Data Engineering*, pages 230–237, 1992.

[SB94] M. Spasojevic and P. Berman. Voting as the optimal static pessimistic scheme for managing replicated data. *IEEE Trans. Parallel and Distributed Sys.*, 5(1):64–73, 1994.

[Tho79] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Sys.*, 4(2):180–209, 1979.

[VMS] DIGITAL OpenVMS systems. `http://www.openvms.digital.com/`.

[vRBM96] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, 1996.