Bulletin of the Technical Committee on

Data Engineering

December 1998 Vol. 21 No. 4 **IEEE Computer Society**

Letters

Special Issue on Data Replication

Conference and Journal Notices

Editorial Board

Editor-in-Chief

David B. Lomet
Microsoft Research
One Microsoft Way, Bldg. 9
Redmond WA 98052-6399
lomet@microsoft.com

Associate Editors

Amr El Abbadi
Dept. of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106-5110

Surajit Chaudhuri
Microsoft Research
One Microsoft Way, Bldg. 9
Redmond WA 98052-6399

Donald Kossmann
Lehrstuhl für Dialogorientierte Systeme
Universität Passau
D-94030 Passau, Germany

Elke Rundensteiner
Computer Science Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering (http: www. is open to all current members of the IEEE Computer Society who are interested in database systems.

The web page for the Data Engineering Bulletin is http://www.research.microsoft.com/research/db/debull. The web page for the TC on Data Engineering is http://www.ccs.neu.edu/groups/IEEE/tcde/index.html.

TC Executive Committee

Chair

Betty Salzberg
College of Computer Science
Northeastern University
Boston, MA 02115
salzberg@ccs.neu.edu

Vice-Chair

Erich J. Neuhold
Director, GMD-IPSI
Dolivostrasse 15
P.O. Box 10 43 26
6100 Darmstadt, Germany

Secretry/Treasurer

Paul Larson
Microsoft Research
One Microsoft Way, Bldg. 9
Redmond WA 98052-6399

SIGMOD Liason

Z.Meral Ozsoyoglu
Computer Eng. and Science Dept.
Case Western Reserve University
Cleveland, Ohio, 44106-7071

Geographic Co-ordinators

Masaru Kitsuregawa (**Asia**)
Institute of Industrial Science
The University of Tokyo
7-22-1 Roppongi Minato-ku
Tokyo 106, Japan

Ron Sacks-Davis (**Australia**)
CITRI
723 Swanston Street
Carlton, Victoria, Australia 3053

Svein-Olaf Hvasshovd (**Europe**)
ClustRa
Westermannsveita 2, N-7011
Trondheim, NORWAY

Distribution

IEEE Computer Society
1730 Massachusetts Avenue
Washington, D.C. 20036-1992
(202) 371-1013
twoods@computer.org

Letter from the Editor-in-Chief

Our New Financial Plan– For the Last Time

The TC on Data Engineering does not generate enough revenue to adequately fund the Bulletin. As TC Chair Betty Salzberg explained it in the June issue, *"we will no longer be automatically mailing out paper copies of the Bulletin to all members of the TC. This will cut down on the expenses of the TC without, we hope, causing undue inconvenience to our members, most of whom will get their copies of the Bulletin through the Bulletin web page."* This information is included in the Bulletin, as hardcopy distribution is our only sure way to reach all TC members. If you are unable to access the Bulletin via our web site and want to continue receiving it, you *must* contact *Tracy Woods, IEEE Computer Society, 1730 Massachusetts Ave., Washington, DC 20036* now. **This issue is the last issue that will be sent in hardcopy to all members.**

The Bulletin and the SIGMOD Anthology

As most of you undoubtedly know, ACM SIGMOD has undertaken to produce an Anthology CDRom containing the complete set of SIGMOD and PODS conference proceedings. The Anthology is being edited by Michael Ley, who has produced a preliminary version for the SIGMOD and DBLP web sites. These sites index all the articles to be included in the anthology. The index plus full articles will appear on the Anthology CD.

The anthology is an initiative of SIGMOD chair Rick Snodgrass, who envisioned a portable, fully-searchable, electronic repository of all relevant database conferences and publications. Currently, the Anthology is slated to include the SIGMOD, PODS, and VLDB proceedings and SIGMOD Record. In addition, the TC on Data Engineering was happy to agree to have the Bulletin (1993-1998) also included. So, even when you are not "on-line", there is no reason to be without full access to these issues of the Bulletin- and of course, to the database material from the ACM and VLDB. I regard the Anthology project as an incredible plus for our community, and applaud the efforts of Rick and Michael in making it a reality. It has been a real pleasure to participate in this collaborative effort of ACM SIGMOD, VLDB and IEEE TCDE and to see the Bulletin included in the anthology.

You can get an early look at the anthology by visiting the SIGMOD web site for it at

http://www.acm.org/sigmod/dblp/db/anthology.html.

This Issue

Database replication has become an increasing exploited technology. It is an important technique for ensuring high availability, and plays an important role in performance of distributed database systems. Commercial database vendors have supported replication for a number of years. Sybase was one of the earliest in this market, and the issue contains an article describing its approach. But replication is by no means a "done deal". It is still an area of active research. These include exploiting replication on the web and using it to improve distributed systems algorithms. And then there is the perennial topic of managing replicas, of how rapidly to propagate changes to replicated data, and how to sort out which replicas need to be read and/or written. I want to thank Amr El Abbadi and Divy Agrawal, who jointly edited the issue, and the authors for producing this interesting and timely issue containing articles addressing these subjects.

David Lomet
Microsoft Research

Letter from the Special Issue Editors

Data replication is one of the oldest and most persistent topics of research in distributed databases. It holds the promise of solving both problems of fault-tolerance and performance. By replicating the data on remote sites, failures may occur and users may still be able to access data; alternatively, if a data item is replicated on a local or nearby site, accesses to that copy are efficient and may involve very low overhead. To obtain these benefits, however, complex and expensive synchronization mechanisms are often needed to maintain the consistency and integrity of data. In fact it has often been argued that due to the synchronization overheads and the increased possibility of deadlock, replication is impractical. Nevertheless, the appeal of replication persists and sometimes even migrates to commercial industrial products. In this special issue various researchers and practitioners will address the current state of replication and its promises.

The first paper, written by Avishai Wool, addresses head-on the promise of replication and asks the question: Is it science or fiction? He explores this question in the context of quorum-based replication and discusses its potential to solve the problems of managing replicated databases. Avishai explains why quorum systems have not fulfilled their old promises, but at the same time argues why new technological advances and applications have brought new requirements that can be addressed by quorums. In fact, he shows that quorum systems may offer a way to scale up throughput in heavily loaded systems.

Many replicated storage systems have advocated relaxed consistency models to get improved availability, scalability and performance. Doug Terry, Karin Petersen, Mike Spreitzer and Marvin Theimer discuss the case for non-transparent replication using the Bayou system. The Bayou system developed at Xerox PARC is an example of a replicated storage system that was designed to strike a balance between application control and complexity. In this paper the authors discuss the tension between overburdening and underpowering applications. They discuss two bayou applications, a calendar manager and a mail reader, and illustrate ways in which they utilize Bayou's features to manage their data in an application-specific manner.

The web is a natural context for replication with its increasing scalability and performance demands. Micha Rabinovich examines various issues that arise in Web content replication, paying special attention to challenges in dynamic replication. The paper starts with an overview of the current state of replication and caching on the web. It then proceeds to offer valuable insights into open questions and research issues both from architectural as well as from algorithmic points of view.

In the next paper, Michel Raynal addresses the problem of atomic commitment in distributed replicated databases. He shows that solutions to the consensus problem can be very useful in developing solutions to the atomic commitment problem. In particular, he develops a solid theoretical foundation for identifying the minimal assumptions a system must satisfy in order to develop non-blocking replica control protocols in spite of process crashes and asynchrony.

Finally, Bob Breton presents the evolution of a replication system, Sybase's Replication Server, and discusses how it supports disaster recovery as well as support for high availability business requirements. In particular, he discusses the need for replication as a means for disaster recovery in real applications that have very stringent availability requirements. In the paper he also discusses a number of alternatives for high availability database architectures and outlines the main risks and benefits of each approach.

Divyakant Agrawal and Amr El Abbadi
UC Santa Barbara

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

Copyright 1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
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.

The Case for Non-transparent Replication: Examples from Bayou

Douglas B. Terry Karin Petersen Mike J. Spreitzer Marvin M. Theimer
Computer Science Laboratory
Xerox Palo Alto Research Center
Palo Alto, CA 94304 USA

Abstract

Applications that rely on replicated data have different requirements for how their data is managed. For example, some applications may require that updates propagate amongst replicas with tight time constraints, whereas other applications may be able to tolerate longer propagation delays. Some applications only require replicas to interoperate with a few centralized replicas for data synchronization purposes, while other applications need communication between arbitrary replicas. Similarly, the type of update conflicts caused by data replication varies amongst applications, and the mechanisms to resolve them differ as well.

The challenge faced by designers of replicated systems is providing the right interface to support cooperation between applications and their data managers. Application programmers do not want to be overburdened by having to deal with issues like propagating updates to replicas and ensuring eventual consistency, but at the same time they want the ability to set up appropriate replication schedules and to control how update conflicts are detected and resolved. The Bayou system was designed to mitigate this tension between overburdening and underempowering applications. This paper looks at two Bayou applications, a calendar manager and a mail reader, and illustrates ways in which they utilize Bayou's features to manage their data in an application-specific manner.

1 Introduction

A major challenge faced by designers of general-purpose replicated storage systems is providing application developers with some control over the replication process without burdening them with aspects of replication that are common to all applications. System models that present applications with ”one-copy equivalence” have been proposed because of their simplicity for the application developer [1, 3]. In particular, the goal of ”replication transparency” is to allow applications that are developed assuming a centralized file system or database to run unchanged on top of a strongly-consistent replicated storage system. Unfortunately, replicated systems guaranteeing strong consistency require substantial mechanisms for concurrency control and multisite atomic transactions, and hence are not suitable for all applications and all operating environments. To get improved levels of availability, scalability, and performance, especially in widely-distributed systems or those with imperfect network connectivity, many replicated storage systems have relaxed their consistency models. For instance, many systems have adopted an ”access-anywhere” model in which applications can read and update any available replica

Copyright 1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

and updates propagate between replicas in a lazy manner [2, 4, 7, 8, 9, 10, 12, 15]. Such models are inherently more difficult for application developers who must cope with varying degrees of consistency between replicas, design schedules and patterns for update propagation, and manage conflicting updates. The harsh reality is that applications must be involved in these difficult issues in order to maximize the benefits that they obtain from replication. The Bayou system developed at Xerox PARC is an example of a replicated storage system that was designed to strike a balance between application control and complexity.

This paper presents both the application-independent and application-tailorable features of Bayou along with the rationale for the division of responsibility between an application and its data managers. Examples drawn from a couple of Bayou applications are used throughout to illustrate how different applications utilize Bayou's features. The applications are a calendar manager and a mail reader. The Bayou Calendar Manager (BCM) stores meetings and other events for individual, group, and meeting-room calendars. A user's calendar may be replicated in any number of places, such as on his office workstation and on a laptop so that he can access it while travelling. Bayou's mail reader, called BXMH, is based on the EXMH mail reader [20]. BXMH receives a user's incoming electronic mail messages, provides facilities for reading messages, and permits the user to permanently store messages in various folders. The BXMH mail database managed by Bayou may be replicated on a number of sites for increased availability or ease of access. Each of these two applications interact with the Bayou system in different ways to manage their replicated data. They demonstrate the need for flexible application programmer interfaces (APIs) to replicated storage systems.

2 Application-independent Features of Bayou

For most replicated storage systems, the basic replication paradigm and associated consistency model are common to all applications supported by the system. While it is conceivable that a replicated storage manager could provide individual applications with a choice between strong and weak data consistency, this made little sense for Bayou. Bayou was designed for an environment with intermittent and variable network connectivity. In such a setting, mechanisms to support strong consistency would not be applicable. Therefore, Bayou's update-anywhere replication model and its reconciliation protocol, which guarantees eventual consistency, are central to the systems architecture. These fundamental design choices over which the application has little or no control are discussed in more detail in the following subsections. Additional application-independent mechanisms for replica creation and retirement are also briefly described. Features that are within an application's control, such as conflict management, are discussed in Section 3.

2.1 Update-anywhere replication model

Bayou manages, on behalf of its client applications, relational databases that can be fully replicated at any number of sites. Each application generally has its own database(s). Application programs, also called "clients", can read from and write to any single replica of a database. Once a replica accepts a write operation, this write is performed locally and propagated to all other replicas via Bayou's pair-wise reconciliation protocol discussed below. This "update-anywhere" replication model, depicted in Figure 1, permits maximum availability since applications can continue to operate even if some replicas are unavailable due to machine failures or network partitions. Thus, it is particularly suitable for applications that operate in mobile computing environments or large internetworks. Because each read and write operation involves a single interaction between a client and a replica, the update-anywhere replication model is also easy to implement and provides good response times for operations.

This replication model was adopted for Bayou because of its flexibility in supporting a diversity of applications, usage patterns, and networking environments [6]. If replicas are intermittently connected, replicas are allowed to arbitrarily diverge until reconciliation is possible. If replicas are few and well-connected, the update-anywhere model is still a satisfactory choice since updates can propagate quickly under such circumstances and

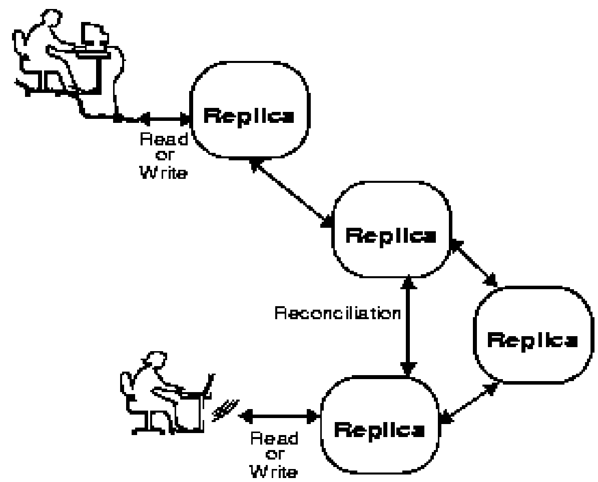

Figure 1: Bayou's update-anywhere replication model.

the replicas remain highly consistent. As described in section 3.1, applications can select reconciliation schedules that best fit their requirements for how much replicas are allowed to diverge.

Consider the example of a user, Alice, managing her personal calendar using BCM. Alice might keep a replica of her calendar on her office machine, one on her laptop, and also one on the office machine of her administrative assistant, Bob, so that her assistant has quick access to her calendar. Alice and Bob's office machines perform reconciliation with each other on a frequent basis so that any updates made to the calendar by either of them are seen by the other with little delay. However, when Alice is travelling, she may update the replica on her laptop while the laptop is disconnected. Any new meetings added by Alice are not readily available to Bob (and vice versa). From her remote destination, Alice occasionally connects to her (or Bob's) office machine via a dial-up modem to exchange recently added meetings, thereby updating their replicas of the shared calendar.

2.2 Reconciliation protocol and eventual consistency

Bayou's reconciliation protocol is the means by which a pair of replicas exchange updates or "writes" [16]. The protocol is incremental so that only writes that are unknown to the receiving replica are transmitted during reconciliation. It requires replicas to maintain an ordered log of the writes that they have accepted from an application client or received from another replica via reconciliation. Pair-wise reconciliation can guarantee that each write eventually propagates to each replica, perhaps by transmission through intermediate replicas, as long as there is an eventual path between a replica that accepts a write and all other replicas. The theory of epidemics indicates that, even if servers choose reconciliation partners randomly, writes will fully propagate with high probability [4]. Arbitrary, non-random, reconciliation schedules can be set up by applications if desired as discussed in section 3.1.

Bayou ensures "eventual consistency" which means that all replicas eventually receive all writes (assuming sufficient network connectivity and reasonable reconciliation schedules) and any two replicas that have received the same set of writes have identical databases. In other words, if applications stopped issuing writes to the database, all replicas would eventually converge to a mutually consistent state. Eventual consistency requires replicas to apply writes to their databases in the same order. Bayou replicas initially order "tentative" writes according to their accept timestamps, and later reorder these writes as necessary based on a global commit order assigned by a primary server [19].

Fortunately, the machinery for managing write-logs, propagating writes, ordering writes, committing writes, rolling back writes, and applying writes to the database are completely handled by the Bayou database managers. Applications simply issue read and write operations and observe the effects of eventual consistency. Applications can optionally request additional session guarantees that affect the observed consistency [18].

2.3 Replica creation and retirement

Bayou permits the number and location of replicas for a database to vary over time. While the replica placement policies are under the control of applications as discussed below in section 3.1, the mechanism for creating new replicas and retiring old ones is application-independent. Bayou allows new replicas to be cloned from any existing replica. The data manager for the new replica contacts an existing replica to get the database schema, creates a local database, and then performs reconciliation with an existing replica to load its database and write-log. Information about the existence of the new replica then propagates to other replicas via the normal reconciliation protocol. This is done by inserting a special ”creation write” for the new replica into the write-log. As this write propagates via reconciliation, others replicas learn of the new replica's existence [16].

Retirement of replicas is similar. A replica can unilaterally decide to retire by inserting a ”retirement write” in its own write-log. The retiring replica can destroy itself after it performs reconciliation with another replica who will then propagate knowledge of the retirement and of other writes that were accepted by the retired replica.

3 Application-tailorable Features of Bayou

In contrast to the mechanisms for update propagation and eventual consistency, policies and functionalities that vary amongst Bayou applications include how they deal with update conflicts, where they place replicas, and which replicas they access for individual operations. Those issues are discussed in this section. Examples taken from the Bayou applications illustrate how different applications can tailor the Bayou system to meet their specific needs.

3.1 Replica placement and reconciliation schedule

The choice of where to place replicas and when replicas should reconcile with each other is an important policy that is under the control of Bayou applications and users. As described above, the mechanism for replica creation is the same for all Bayou applications. However, the choice of the time at which a replica gets created and the machine on which it resides is completely determined by users or system administrators. The only condition for a replica to be successfully created is that one other replica be available over the network.

Similarly, since Bayou's weak consistency replication model does not require updates to be immediately propagated to each replica, users are afforded a large amount of flexibility in setting up reconciliation schedules. Experience suggests that such schedules are generally dictated more by the user's work habits than by the needs of a particular application. For example, a user who works from home in the evening, may wish his office workstation to reconcile with his home machine at 5:00 pm each evening, but does not care about keeping his home machine up-to-date during the day. Also, users and applications often know when are good times or bad times to reconcile with another replica. For instance if the application is in the process of doing a number of updates or refreshing its display, it may not want the database to change underneath it. As another example, a travelling user may dial-in from a hotel room and want reconciliation with the office performed immediately rather than waiting for the next scheduled time.

3.2 Replica selection

Bayou applications generally issue read and write requests without even being aware of which replicas they are accessing. The Bayou client library, which implements the application programming interface (API) and is loaded with the application code, chooses an appropriate replica on which to perform a read or write operation. This choice is based on the availability of replicas, cost of accessing them, and application-chosen session guarantees. The Bayou client library automatically adapts to changing network conditions and replica availability.

Originally, Bayou provided no ability for an application to override the replica selections made by the client library. That is, a Bayou application could not direct its operations to a particular replica. We presumed that most applications, while concerned with the consistency of the data they read, do not wish to be concerned with the specifics of which replicas to access. Moreover, we reasoned that applications do not have enough information about the underlying network connectivity or communication costs to make reasonable decisions about replica selection. What we failed to recognize initially is that users do, in fact, often know quite a bit about the network characteristics as well as the capabilities and consistency of various replicas. For instance, Alice might prefer to access the copy of her calendar that resides on her workstation rather than the one on her laptop, even if the calendar client application is running on the laptop and both the workstation and laptop replicas are available. Hence, users occasionally do want to provide hints about which replicas to access.

Also, there are situations in which an application may want control over replica selection. For instance, an application that supports synchronous collaboration between a number of users, such as a shared drawing tool, may want all these users to access the same replica so that they share the exact same database state. Replication may be desired by this application solely for fault-tolerance, that is, so that it can fail-over to a secondary replica in case the primary fails. Thus, in the second implementation of the Bayou system, we added support for application-controlled replica selection.

3.3 Conflict detection

An inherent feature of Bayou's update-anywhere replication model is that concurrent, conflicting updates may be made by users interacting with different replicas of a shared database. For instance, in the Bayou Calendar Manager (BCM), Alice and Bob could schedule different meetings for Alice at the same time. Such conflicts must be dealt with by each application in an application-specific manner.

The definition of what constitutes a conflict varies from application to application and potentially from user to user. Traditionally, database managers and file systems have pessimistically treated any concurrent writes as conflicting. However, experience with Bayou applications suggest that not all concurrent writes result in application level conflicts. Moreover, writes to separate tuples, which are traditionally viewed as independent, may, in fact, conflict according to the application. Consider BCM which stores each calendar entry or meeting as a separate tuple in the database. Without help from the application, the storage system would detect conflicts as operations that are concurrent at the granularity of either the whole database or individual tuples. If the former, then any concurrently added meetings would be detected as conflicting; if the latter, then no meetings would ever conflict since they involve updating different tuples. Neither of these cases reflect BCM's semantic definition of a conflict.

In BCM, two writes that add new meetings to a calendar or modify existing meetings conflict if their meetings overlap in time and involve the same user(s) or conference room. This simple definition of conflicts is readily supported by Bayou's application-specific conflict detection mechanism. However, we discovered in practice that it did not satisfy all BCM users; some users would prefer to allow overlapping meetings not to conflict and have them scheduled on their personal calendar so they can decide later which meeting to actually attend.

BXMH has a much more complicated model of conflicting operations on a shared mailbox. While BCM basically has a single type of conflict, BXMH has dozens of potential conflict scenarios. BXMH supports 13 operations on a mailbox: adding a new message, moving a message to a named folder, creating a new folder, renaming a folder, deleting a message, and so on. Each of these operations can conflict with other operations in various ways. Moreover, when designing this application, we discovered that potential users could not always agree on which operations conflict under what conditions. The result is that BXMH, through its ”conflict preferences menu”, allows its users to decide what types of concurrent operations should be considered conflicting. Figure 2 shows one of the many conflict scenarios that appears on the BXMH conflict preferences menu. In this example, the user is asked to decide whether moving a message from one folder to another conflicts with a concurrent operation that renamed the destination folder and, if so, how the conflict should be resolved.

Figure 2: Sample conflict scenario from BXMH's conflict preferences menu.

Although BCM and BXMH have very different notions of conflicting operations, they both rely on the same mechanism to detect their conflicts, namely Bayou's dependency checks [19]. A dependency check accompanies each write performed by an application. The dependency check is a way for the application issuing the write to detect whether the write conflicts with other concurrent writes. Specifically, a dependency check is a query (or set of queries) and a set of expected results. When the dependency query is run at some replica against its current database and returns something other than the expected results, the replica has detected a conflict; in this case, the replica resolves the conflict, as discussed below, rather than performing the given write. Observe that dependency checks are often specific not only to the application but also to the particular write operation.

For example, if Alice adds a meeting to her calendar from 11 to noon on Friday, BCM creates a dependency check for this write that queries the database for other calendar entries at this time and expects none. Bob might concurrently add a conflicting meeting, say at 11:30 on Friday, because his replica has not yet received Alice's write. If Bob's write is ordered before Alice's, then the dependency check included in Alice's write will fail.

3.4 Conflict resolution

Strategies for resolving detected conflicts also vary from application to application and user to user. In BCM, a conflict involving two meetings is resolved by trying to reschedule one of the meetings. The meeting that was added last according to Bayou's write ordering is the one that is rescheduled. In BXMH, the resolution depends on the type of conflict and on the user's preferences. For example, a user might choose to resolve the conflict in Figure 2 by moving the message to the renamed folder, by leaving the message in its original folder, by creating a new folder for the message or by moving the message to some other existing folder.

Merge procedures in Bayou are the means by which applications resolve conflicts. Specifically, each Bayou write operation actually consists of three components: a nominal update, a dependency check, and a merge procedure [19]. The nominal update indicates changes that should be made to the application database assuming that no conflicting writes have been issued. The dependency check, as discussed above, detects conflicts involving the write. The merge procedure is a piece of application code that travels with the write and is executed to resolve any detected conflicts. The merge procedure associated with a write can query the executing replica's database and produces a new update to be performed instead of the nominal update. Since merge procedures are arbitrary code, they can embody an unlimited set of application-specific policies for resolving conflicts.

An application is free to pass null dependency checks and merge procedures with each write, in which case the writes issued by the application resemble normal database updates. Importantly, even in the application ignores conflicts, its database is guaranteed to eventually converge to a consistent state at all replicas. Concurrent updates may cause the application not to see some updates because they are overwritten, but eventual consistency is preserved.

3.5 Reading tentative data

Bayou gives applications the choice of reading only committed data or data that may be in a tentative state. The rationale was that some applications may only want to deal with data after it has been committed. Interestingly, the Bayou applications that have been built to date never select the commit-only option when reading data. This is because users always want to see updates that they have made, even if the update has not yet been committed. Bayou indicates which data items an application reads are tentative and which are committed. How the application deals with the information varies with the application. BCM uses this information to show tentatively scheduled meetings in a different color than committed ones. The expectation is that a committed meeting is not likely to change in time, at least not without the meeting organizer informing the participants explicitly, while tentative meetings could get rescheduled due to conflicts. So it is important for the user to know which meetings are tentative and which are not. BXMH, on the other hand, does not distinguish between tentative and committed data when showing a folder's content to the user. The user does not really care whether a particular message is tentatively in a folder as long as the message is successfully displayed when the user clicks on it.

4 Related Work

Early weakly-consistent replicated systems, like Grapevine [2], were intimately tied to particular applications, like electronic mail and name services. The issue of designing replicated storage systems that effectively support a number of diverse applications arose when replication was added to conventional file systems and database management systems. Many of these systems started with the goal of replication transparency but gradually ended up adding hooks for applications to give input to the replication process.

Replicated file systems like Coda [11] and Ficus [17] present applications with a traditional file system interface but also allow them to install "application-specific resolvers" to deal with conflicting file updates. Coda has also recently added "translucent caching" which hides some caching details from users and applications while revealing others [5, 14].

In the database arena, Oracle 7 supports weakly consistent replication between a master and secondary replicas or between multiple masters. It permits applications, when specifying their database schema, to select rules for resolving concurrent updates to columns of a table; each "column group" can have its own conflict resolution method [15]. Applications can provide a custom resolution procedure or choose from a set of standard resolvers.

Lotus Notes, like Bayou, utilizes pair-wise reconciliation between replicas and allows its system administrators to specify arbitrary replication schedules [13]. Notes also permits users and applications to manually invoke reconciliation between two replicas. It detects conflicting updates to a document using timestamps, but has no support for application-specific conflict resolution; alternative versions can be maintained for documents that are concurrently updated.

Bayou, since it was not concerned about backwards compatibility or supporting existing applications, could design a new API that permits more direct application control over various aspects of replication and consistency management. Bayou's conflict resolution mechanism, based on per-write merge procedures, is more flexible than that of other systems, as is its support for application-specific conflict detection.

5 Conclusions

Designing an application programming interface (API) for replicated data is difficult since one must balance the desire for simplicity against the amount of control afforded the application. Simplicity argues for placing common functionality in the replicated storage system, for presenting a storage model that is as close as possible to that of a non-replicated system, and for minimizing aspects of the underlying replication state that are exposed to the application. However, to obtain the maximum benefits from replication, an application needs methods for cooperating with the replicated storage system in the management of the application's data. Permitting such cooperation without requiring the application to assume unnecessary responsibility for the replication process is the key challenge.

The development of Bayou and its applications has allowed us to explore these issues of interaction between applications and replicated data managers. In Bayou, data managers take full responsibility for propagating and ordering updates and ensuring that replicas converge to a consistent state, while applications may control the detection and resolution of update conflicts, create and destroy replicas at convenient times, and set up reconciliation schedules.

Experience building a number of Bayou applications has confirmed the belief that applications need customized control over the replication process. The two applications used as examples in this paper, a calendar manager and a mail reader, have very different policies for detecting and resolving update conflicts. Additionally, they often want different reconciliation schedules. Interestingly, these choices vary not only between applications but also among users of the same application. We conclude that "replication transparency", while a laudable goal for supporting legacy applications, is not appropriate for a replicated storage system intended to support a number of applications in diverse networking environments.

6 Acknowledgments

We are grateful for the contributions of our colleagues and interns who have aided in the design and implementation of Bayou and its applications including: Atul Adya, Surendar Chandra, Alan Demers, Keith Edwards, Carl Hauser, Anthony LaMarca, Beth Mynatt, Eric Tilton, Brent Welch, and Xinhua Zhao.

7 References

References

[1] 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.

[2] A. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine: An exercise in distributed computing. Communications of the ACM 25(4):260-274, April 1982.

[3] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in a partitioned network: A survey. ACM Computing Surveys 17(3):341-370, September 1985.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. Proceedings Sixth Symposium on Principles of Distributed Computing, Vancouver, B.C., Canada, August 1987, pages 1-12.

[5] M. R. Ebling. Translucent cache management for mobile computing. Carnegie Mellon University technical report CMU-CS-98-116, March 1998.

[6] W. K. Edwards, E. D. Mynatt, K. Petersen, M. J. Spreitzer, D. B. Terry, and M. M. Theimer. Designing and Implementing Asynchronous Collaborative Applications with Bayou. Proceedings User Interface Systems and Technology, Banff, Canada, October 1997, pages 119-128.

[7] R. A. Golding, A weak-consistency architecture for distributed information services, Computing Systems, 5(4):379-405, Fall 1992.

[8] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. Proceedings 1996 ACM SIGMOD Conference, Montreal, Canada, June 1996, pages 173-182.

[9] R. G. Guy, J.S. Heidemann, W. Mak, T.W. Page, Jr., G.J. Popek, and D. Rothmeier. Implementation of the Ficus replicated file system. Proceedings Summer USENIX Conference, June 1990, pages 63-71.

[10] L. Kalwell Jr., S. Beckhardt, T. Halvorsen, R. Ozzie, and I. Greif. Replicated document management in a group communication system. In Groupware: Software for Computer-Supported Cooperative Work, edited by D. Marca and G. Bock, IEEE Computer Society Press, 1992, pages 226-235.

[11] P. Kumar and M. Satyanarayanan. Supporting application-specific resolution in an optimistically replicated file system. Proceedings IEEE Workshop on Workstation Operating Systems, Napa, California, October 1993, pages 66-70.

[12] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing high availability using lazy replication. ACM Transactions on Computer Systems 10(4):360-391, November 1992.

[13] R. Larson-Hughes and H. J. Skalle. Lotus Notes Application Development. Prentice Hall, 1995.

[14] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. Proceedings Fifteenth ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, December 1995, pages 143-155.

[15] Oracle Corporation. Oracle7 Server Distributed Systems: Replicated Data, Release 7.1. Part No. A21903-2, 1995.

[16] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible Update Propagation for Weakly Consistent Replication. Proceedings 16th ACM Symposium on Operating Systems Principles, Saint-Malo, France, October 1997, pages 288-301.

[17] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. Popek. Resolving file conflicts in the Ficus file system. Proceedings Summer USENIX Conference, June 1994, pages 183-195.

[18] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer and B. B. Welch. Session guarantees for weakly consistent replicated data. Proceedings Third International Conference on Parallel and Distributed Information Systems, Austin, Texas, September 1994, pages 140-149.

[19] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. Proceedings Fifteenth ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, December 1995, pages 172-183.

[20] B. B. Welch. Customization and flexibility in the exmh mail user interface. Proceedings Tcl/Tk Workshop, Toronto, Canada, 1995, pages 261-268.

Issues in Web Content Replication

Michael Rabinovich
AT&T Labs
180 Park Avenue
Florham Park, NJ 07932
misha@research.att.com

Abstract

Web content replication began as explicit manual mirroring of Web sites. It has since evolved into user-transparent request distribution among a static set of mirrors. The next step (especially important for large-scale Web hosting service providers) is dynamic content replication, where object replicas are created, deleted, or migrated among Web hosts in response to changing usage patterns. This paper examines issues that arise in Web content replication, paying special attention to challenges in dynamic replication.

1 Introduction

As commercial interest to the Internet continues to grow, the issues of scalability and performance become increasingly important. According to UUNET, the largest Internet Service Provider (ISP), its network traffic has been doubling every 100 days [12]. In fact, we are on the verge of another qualitative jump in Internet load levels, due to the upcoming replacement of slow modem lines, which act as floodgates limiting user access to the Internet, with much faster alternatives like ISDN lines and cable modems. It is unlikely that the brute-force approach of adding ever increasing network and server capacities would solve the Web scalability problem in a foreseeable future. Consequently, caching and replication, being the primary tools that address these issues, are fast becoming a focus of attention in both industrial and academic research communities. This paper examines issues in replication on the Internet and outlines research directions that are both interesting and practically important.

One can distinguish two main environments where replication can be used. One is a stand-alone replicated Web site; the other is a platform for a large-scale hosting service that hosts Web objects from third-party information providers. In the limited-scale stand-alone environment, a brute-force worst case resource allocation can be feasible. Manual monitoring of demand and decisions on the number and placement of replicas is feasible as well. Thus, a stand-alone Web site can often use static replication (or *mirroring* in the Internet parlance).

In the hosting service environment, the waste of the worst-case design would be multiplied by potentially millions of hosted Web sites; the decision space for replica placement could also become unmanageable. Thus, dynamic automated replication is needed. According to Probe Research [14], the market for hosting services will reach $2.5 billion by 2001. This gives an extra imperative to develop appropriate technologies for automatic dynamic replication of Web objects.

Copyright 1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In general, there are two fundamental questions any replicated system must deal with. One is assigning requests to replicas. The other is deciding on the number and placement of replicas for all objects. These issues are often studied in isolation, as a server selection problem and a file allocation problem. However, they strongly influence one another, and in a system with dynamic replication must be studied together.

2 Caching vs. Replication

Being an immature field, the Web has created many terms to denote similar things. In particular, there is some confusion between caching and replication. To develop an intuition for the difference between the two, I propose the following distinction.

Caching is storing an object at a place that sees the object anyway in the course of request processing. Examples include browser cache, proxy cache, server main memory cache. Replication is storing an object at a place that would not otherwise see the object. Web site mirroring provides an example of replication.

Alternatively, one can say that the difference between a cache and a replica is that a cache routinely sees both hit and miss requests, while a replica normally sees only hits (except when a request arrives for an object that has been deleted). In other words, requests flow to the cache regardless of the cache contents, while requests arrive at a replicated server only if that server is believed to have a replica of the requested object.[1]

While useful and necessary, caching itself cannot solve the scalability problem of the Web. Indeed, many objects are not cacheable but replicable. These include dynamic objects with read-only access and personalized objects ("cookied" requests). In addition, many objects that are updated as a result of accesses are still replicable because updates commute. A typical example is provided by so called hit metering, where an object maintains a counter of times it has been accessed.

Another advantage of replication is that it is under full control of the service provider. Thus, issues like copyright enforcement, hit metering, data consistency are much easier. Reluctance to deal with these issues leads sometimes to "cache busting" by content providers, who use various available mechanisms to prevent caching of their pages. Finally, replicated servers are free to use proprietary protocols that would be more efficient for moving object replicas between servers or maintaining them consistent, while caches are confined to standards.

3 Mechanisms for Web Request Redirection

Transparent replication entails redirecting a client's request for a logical object to one of the physical object replicas. To be grounded in reality, any proposal for replication on the Web, must explain the assumed mechanism for such redirection. Understanding this issue lets one distinguish between approaches that are deployable in practice from more speculative research ideas (this is not to say that the latter are not valuable).

Accessing a Web resource, e.g., `www.foo.com/home.html`, involve the following main steps.

1. The client sends the domain name of the Web site (`www.foo.com`) to its local domain name *resolver*. The resolver obtains the corresponding IP address by contacting a well-known root of the domain name server (DNS) hierarchy and ultimately querying foo's DNS server. The resolver then returns this IP address to the client.

2. The client sends the request for `home.html` to the received IP address. This request follows a router path on the Internet until it arrives at the Web server, which sends back the requested page.

Let us consider points on the request processing path where the request can be transparently redirected to one of several replicas of the object, and the tradeoffs involved. (We will not consider non-transparent replication where users are asked to choose among multiple URLs of the object.)

[1] Note that replication in the above sense is often called "push caching" or active caching. I find these terms confusing and will not use them here.

3.1 Client Multiplexing

In this approach, the client (Web browser or a proxy server) obtains a set of physical replicas and chooses the one to send its request to. Three main mechanisms fall into this category.

In the first approach, the DNS server of the service provider returns the IP addresses of servers holding a replica of the object. The client's DNS resolver chooses a server among these. To decide, the resolver may issue probes to the servers and choose based on response times to these probes, or it may collect reports from the the client on performance of passed accesses to these servers. This approach is used by Bhattacharjee et al [5] and by Beck and Moore [4]. Its advantage is that no extra communication is added to the critical path of request processing. There are also several shortcomings. First, the approach relies on clients using a customized DNS resolver. If this resolver relies on client performance reports, then the client (i.e., browser or proxy) software must be modified as well. Second, the DNS infrastructure relies heavily on DNS response caching to cope with its load. Therefore, replica sets cannot be changed frequently without the danger of resolvers using stale replica sets. At the same time, reducing the caching time may just move the scalability bottleneck to the DNS infrastructure, provided enough clients adopt the approach. Thus, this approach is most appropriate for static replication.

The second approach relies on Java applets to perform replica selection on the client [21]. The URL of an object actually points to a Java applet, which embeds the knowledge about the current replica set and the procedure for replica selection. This approach requires no changes to clients. However, unlike the previous approach, it involves an extra TCP communication to download the applet.

The third approach, proposed by Baentsch et al [3], propagates information about replica sets in HTTP headers. It requires changes to both Web servers and clients (proxy servers in this case) to process extra headers. Clients must also be modified to implement replica selection.

3.2 IP Multiplexing

A plethora of commercial products offers multiplexing routers. A now-outdated survey can be found at [9]. In this approach, a special multiplexing router (or a multiplexor) is placed in front of a server farm. The domain name of the Web site is mapped to the IP address of this router, so all clients send their packets to it. When the first packet from a client arrives, the multiplexor selects a server in the farm and forwards the packet (and all subsequent packets in the same session) to this server. The multiplexor also intercepts the response packets and modifies their headers to contain its IP address rather than that of the server. The multiplexor maintains a *sessions database*, which records a server chosen for each active communication session. The sessions database is needed to ensure that, once a server is chosen for servicing a request, all packets from this client in this communication session go the same server.

This approach uses standard clients and DNS and Web servers. However, by having a focal communication point, it does not scale geographically. Thus, it is mostly used for load sharing in a server farm on a local-area network.

An interesting twist on this approach is provided by IBM Network Dispatcher [11]. Its multiplexor selects servers as before. But servers respond to clients directly, bypassing the multiplexor and using the multiplexor's IP address as the sender address. Since most of the data flows from servers to clients, this provides better geographical scalability, especially from the bandwidth consumption perspective. The latency gain is limited by the need for client acknowledgements to travel via the multiplexor.

3.3 DNS Indirection

Several domain name server implementations (e.g., the one used in CISCO Distributed Director [6]) allow the Web site's DNS server to map a host domain name to a set of IP addresses and choose one of them for every client query, based on such factors as the query origin and the load of server replicas. The difference with DNS-based client multiplexing is that choosing a replica occurs at the Web site, not at the client's DNS resolver. This approach can scale well geographically. Unfortunately, DNS response caching by clients complicates changing replica sets and controlling request distribution among replicas. At the same time, reducing the lifetime of cached DNS responses may shift the performance bottleneck to the DNS infrastructure.

In general, DNS system was designed for mostly an append-only database existing mappings between a host name and an IP address rarely ever changes. In dynamic replication schemes, however, an existing mapping changes with every change in the set of replicas for a given object.

3.4 HTTP Redirection

HTTP protocols allows a Web server to respond to a client request with a special message that tells the client to resubmit its request to another server. This mechanism can be used to build a special Web server which accepts client requests, chooses replicas for them and redirects clients to these replicas. Commercially, CISCO Distributed Director [6] and WindDance Web Challenger [19] implement this functionality.

An advantage of HTTP-level redirection is that replication can be managed at fine granularity, down to individual Web pages, whereas other mechanisms postulate the entire Web site as the granule. A disadvantage is that this mechanism is quite heavy-weight. Not only does it introduce an extra message round-trip into request processing, but also this round-trip is done over HTTP, which uses the expensive TCP protocol as the transport layer.

3.5 Anycast

In the next generation of the Internet routing protocol (IPv6), an *anycast* service [13] will be supported. Without getting into the details of the mechanism, this service assumes that the same IP address is assigned to a set of hosts, and each IP router has in its routing table a path to the host that is the closest to this router. Thus, different IP routers have paths to different hosts with the same IP address.

There is virtually no overhead for request indirection with anycast, since packets travel along router paths anyway. However, it assumes very static replica sets (since a change in the replica set would take long time to reflect in router tables throughout the Internet) and a rigid request distribution (because all requests from a given client normally go to the same host, chosen based on the network proximity). Advances in research on *active networks* [17], where applications can inject ad hoc programs into network routers, may alleviate the last limitation.

4 Research Issues - Architecture

We have seen that most existing architectures for request indirection lack the flexibility needed for dynamic replication. HTTP-based redirection might be suitable, but seems too heavy-weight for most cases. Thus, supporting dynamic replication requires a special highly updateable name service. Such a *Web name service* would maintain a *mapping database* that records logical object names and the names of their corresponding physical replicas, perhaps along with some prior replica usage or performance statistics. The Web name service would use this information to resolve requests for logical objects into requests to object replicas.

An architecture of such service is an open issue. There are a variety of questions to be resolved. The first question is whether the intended use is to be an Internet-wide service or specific to particular sites or hosting platforms. A general Internet-wide name service could be a logical outgrowth of the Uniform Resource Name (URN) proposal whose goals included adding location independence to the Web naming scheme [10]. Unfortunately, the URN effort was too ambitious for any consensus to emerge, and it has not caught up so far. Reviving a slimmed-down URN effort, that would concentrate only on location management, would be beneficial for the Web. A name service for an individual Web site would have very different architecture than a name service for a hosting platform, due to the differences in scale and perspective (e.g., backbone traffic reduction is a major concern for a hosting platform but is of peripheral interest to a Web site).

To be a little more specific, let us consider a hosting service provided by an Internet Service Provider (ISP). The main components of such a system are Web hosting servers (hosts for short), the backbone, and gateway routers through which the platform communicates with the rest of the Internet and with the subscribers of the ISP. Since gateway routers see all client requests anyway, they are a convenient place to redirect requests to various hosts that currently have a replica of requested objects. Redirection at these points could be achieved by co-locating Web name servers on the some LAN with the gateway routers. Requests could be diverted to these name servers by resolving the DNS name of any hosted Web site to the IP address of the name server that is the closest to the requester. Alternatively, it could be done by anycast when available. Note that we cannot rely on the gateway routers themselves to divert requests to their local Web name servers, since there is no guarantee that all packets comprising the request will come to the same gateway router.

Many important issues need be resolved within this framework. Since we have a replica of the name server co-located with every gateway router, the number of name server replicas may be high. However, the frequency of updates to the name server database limits the number of replicas, since updating a large number of replicas is impractical. In fact, some request distribution algorithms (e.g., round-robin) rely on prior replica usage information, which they record in the mapping database and update on every access. This would make name server replication even more problematic. A possible approach to resolving this dilemma is outlined in [16].

Another important decision is to choose between *iterative* and *recursive* request processing. A name server can either return the physical replica ID to the client and let the client access the replica directly, or it can obtain and forward the document to the client. The first alternative (which we call iterative, in analogy with the DNS terminology), reduces the load on the name server and allows clients to cache name resolutions for future use. The second approach, called recursive, hides the identity (and even existence) of hosts from the clients and eliminates an extra message exchange with the client.

One more area of interest is caching within this architecture. In the iterative request processing approach, clients might cache name resolutions for future use. What should be the policies for caching name resolutions? How to enforce these policies? What is the influence of such caching on request distribution and replica placement decisions? Also, one could further reduce backbone traffic by caching especially hot documents at the name servers, at the expense of increasing their load. What are the right policies in this respect?

5 Research Issues - Algorithms

As already mentioned, every replicated system must deal with two fundamental issues - distributing requests to object replicas and deciding on the number and placement of replicas.

5.1 Request Distribution

We already discussed architectural issues in supporting request distribution in Section 4. Let us now consider algorithms these architectures might run. One can distinguish two main classes of algorithms for request distribution - feedback and non-feedback algorithms [1]. Feedback algorithms rely on some aggregate metrics obtained from the hosts, e.g., in the form of load reports. Non-feedback algorithms use only information that name servers obtain from processing prior requests, such as the number of times they assigned previous requests to various replicas. Examples of non-feedback algorithms include the round-robin request distribution or always choosing the closest replica for each request.

Feedback algorithms in general can achieve better request distribution. For instance, a feedback algorithm can take a weighted average of server load and the client-server network distance as a criterion for choosing the replica for a request. On the other hand, feedback schemes make it difficult to predict the effects of an object migration or replication on request distribution. Indeed, request distribution for an object becomes dependent on the popularity of many other objects co-located at the same servers, so moving one object may affect request distribution for other objects and other servers.

Non-feedback algorithms distribute requests "blindly" and are necessarily suboptimal. At the same time, they de-couple distribution of requests to different objects and make estimating the effects of object relocations feasible. As a simple example, creating a second replica of an object in the round-robin scheme will reduce the request rate for the existing replica by half, leaving the request rate for all other objects the same.

Of course, round-robin and always-the-closest are too simplistic algorithms: the former is oblivious to network proximity and is not useful in reducing bandwidth consumption; the latter disregards the load and will continue to send all requests to an overloaded server no matter how many additional replicas are created, if the requests come from nearby clients. A more sophisticated non-feedback algorithm can combine the rank of replicas in the round-robin order with the proximity rank to account for both server proximity and load, while retaining the ability to predict the effects of replication. We described one such algorithm in [16]. However, much more work needs to be done. In fact, it appears that the best choice is to use a feedback algorithm when the overall load in the system is low and switch to non-feedback during high-load periods [1].

5.2 Replica Placement

Another important problem is deciding how many replicas of various objects to have and where in the network to place them. We call this a *replica placement* problem. There are number of issues to be resolved.

The most important issue to resolve is the choice between static and dynamic replica placement. Static replica placement assumes very coarse-granularity decisions that typically involve a systems administrator, based on the observed access characteristics. With dynamic replica placement, the system constantly monitors access to various resources and continuously adjusts replica sets for all objects. The choice between these two approaches is not that obvious. Static placement imposes less stringent requirements for the request distribution algorithm, because the latter does not have to allow predicting effects of individual object replication or migration. So, a more efficient request distribution algorithm can be used, which, for the same assignment of replicas to hosts, would provide better traffic and latency reduction. On the other hand, static placement is error-prone, and can become infeasible for a large-scale system. Moreover, it does not adjust to changes in access patterns or environment (e.g., Internet topology). Overall, static placement may be a feasible solution for a large stand-alone Web site, while dynamic placement seems essential for a *hosting service*, a platform that hosts Web resources from third party content providers. Indeed, a scale of a hosting platform can potentially make worst-case resource allocation of static approaches too wasteful, and the search space for placing replicas too large for a static solution to be feasible.

Within the static approach, the main issue becomes choosing a good request distribution algorithm (usually referred to as server selection in this context), which in this case would most likely be feedback-based. Server selection can use many heuristics, sometimes based on contradicting metrics. The problem here is choosing good metrics and combining them to arrive at a compromise strategy for server selection. We elaborate more on this issue in Section 5.3.

For the dynamic approach, request distribution must be considered in combination with the algorithm for replica placement. The combination must consider load and proximity factors and provide acceptable *responsiveness* in adjusting to changes in access patterns. The responsiveness can only be achieved if objects are relocated *en masse*, without waiting to observe effects after each move. This entails being able to predict (or at least bound) the effects of individual object migrations or replications on request distribution, so that placement decisions can be made for multiple objects at once, without waiting to observe the effects of each move.

Another issue related to dynamic replica placement is the *granularity* of replication. At the finest granularity, the placement of each Web page can be considered separately. The other extreme is to consider placement of entire Web sites. Various intermediate object groupings are also possible. Finer granularity imposes higher overhead for collecting and maintaining access statistics and for placement decisions. Coarser granularity may increase the overhead for transferring objects between sites. In some cases of dynamically generated pages, a group of files, executables, and other environment state *must* be migrated together, since they are used together for page generation. The cost of tranferring such bundled objects may be high and should be taken into account in a replica placement algorithm. A language for specifying these bundled objects must be designed. Marimba's DRP protocol is an example of work in this direction [8]. In addition to manual bundling of objects, some objects could conceivably be bundled together automatically based on similarity of access patterns. The policies for (and the feasibility of) doing this is another issue for future research.

There is also an issue of consistency between replica sets and name service database which maps logical object IDs to sets of physical replicas. Changes in replica sets must be reflected in the name service. If the mapping database of the name service does not agree with actual replica sets, requests may be assigned to non-existent object replicas. On the other hand, migrating an object and updating the mapping database within a distributed transaction would interfere with processing client requests by the name service, and is therefore undesirable. A better approach is to ensure that the replica set specified in the mapping database be always a subset of the actual replica set. In other words, when a replica is to be deleted, it is first deleted from the mapping database, and only then physically removed from the hosting server. When a replica is to be created, it is added to the mapping database only after being successfully created by the hosting server. Migration can be treated as replica creation followed by replica deletion.

An additional complication is that the name service might have previously directed some requests to the replica being deleted, and these requests have not yet reached the replica. This situation is especially dangerous iterative-style name service, with potentially longer delay between request assignment and request arrival at the replica. A protocol must be provided for tracking the number of outstanding requests to each physical replica of each object and for ensuring that a replica is physically deleted only after all such requests have been serviced.

5.3 Replica Ranking

Request distribution algorithms rely on some metrics to discriminate among replicas. Numerous different metrics have been proposed. Among them, there are Web server load, the latency of a ping message, the response time of prior Web document fetches, the latency of prior Web document fetches (the difference being that the latter measures time until the first portion of the data arrives), the network distance (in hops), the geographical distance (in miles). Some of these metrics postulate a place where request redirection takes place. For instance, latency and response time metrics depend on network congestion between a host and the requesting client. Thus, they make sense for a point that lies on the path from the host to the client. As an example, the DNS server of a Web site of hosting service cannot use ping latencies for choosing replicas because these latencies will measure network congestion between replicas and the DNS server, not between replicas and the requesting client. Other metrics, such as host load and distance can be used irrespective of the indirection point. Some metrics, like server load, require a feedback-based algorithm and complicate prediction of effects of changing a replica set on request distribution [1]. Finally, in most cases multiple metrics must be combined in some way, e.g., network distance and server load, in ranking replicas. Choosing a proper metric influences the architecture and algorithm choices and has a profound effect on performance.

5.4 Replica Consistency

This is the area where much synergy exists with existing database research. All Web objects can be divided into the following types:

1. Objects that do not change as the result of user access. These objects can be either static HTML pages, or dynamic pages that retrieve information, i.e., weather reports, or map drawing services.

2. Objects in which the only per-access modification involves collecting access statistics or performing other commuting updates.

3. Objects that be updated as a result of user accesses, and where these updates do not commute.

A majority of Web accesses are to static objects, which can only be modified by their authors but not user accesses. The rate of author modifications is negligible compared to the access rate, so it should not play role in replica placement decisions. Replica consistency can be maintained by using the primary copy approach. Depending on the needs of the application, updates can propagate from the primary asynchronously to the rest of currently existing replicas either immediately or in batches using epidemic mechanisms [7]. There are many ideas in epidemic replication research that are directly applicable to this application, such as determining efficiently which pages need to be copied [15], ensuring the consistency of propagation of multi-page updates [2], and that any given client sees monotonically increasing object versions [18]. One technical issue that must be resolved is how the primary copy is chosen, and what guarantees (if any) are provided to avoid choosing multiple primaries by multiple updaters.

Objects in the second category can still be freely replicated if a mechanism is provided for merging updates recorded by different replicas. The problem arises if content served to clients includes updateable fields, as is the case of hit counts appearing in some pages. If the application requires this information to be always current, then such objects become equivalent to objects in the third category for the purpose of replication.

Objects in the third category can be freely migrated, but their replication strategy must depend on the mix of read-only and update accesses. While some ideas from database research are clearly relevant to this problem (e.g., [20]), Web specifics require more work in this area.

6 Conclusions

This paper describes design space and research directions for content replication on the Web. Content replication is extremely important for Web scalability. Most work to date has concentrated on server selection algorithms assuming a fixed replica set. While much work still remains to be done in this area, the next big step is exploring dynamic replication and migration schemes.

Dynamic replication will be especially important for providers of *hosting services*, which maintain platforms for hosting Web content owned by third parties. As the scale of these systems grows, static manual mirroring will become increasingly wasteful and intractable. Consequently, dynamic replication will be one of enabling technologies for these systems. A plethora of issues must be resolved in the dynamic replication context, including request distribution and replica placement algorithms, granularity of replication, combining replica rankings according to various metrics, and consistency of control information in the system.

Acknowledgements

I wish to thank Theodore Johnson and H. V. Jagadish for reading a draft of this article and their useful comments.

References

[1] A. Aggarwal and M. Rabinovich. Performance of replication schemes on the Internet. Technical Report HA6177000-981030-01-TM, AT&T Labs, October 1998. Also available as http://www.research.att.com/~misha/radar/tm-perf.ps.gz.

[2] D. Agrawal, A. El Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '97)*, pages 161–172, May 1997.

[3] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Enhancing the web infrastructure – from caching to replication. *IEEE Internet Computing*, pages 18–27, Mar-Apr 1997. Also available at http://neumann.computer.org/ic/books/ic1997/pdf/w2018.pdf.

[4] Micah Beck and Terry Moore. The Internet-2 distributed storage infrastructure project: An architecture for internet content channels. In *3rd Int. WWW Caching Workshop*, Manchester, UK, June 1998. Available at http://wwwcache.ja.net/events/workshop/18/mbeck2.html.

[5] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei. Application-layer anycasting. In *INFOCOM*, 1997.

[6] Cisco Systems, Inc. DistributedDirector. White paper. http://www.cisco.com/warp/public/734/distdir/dd_wp.htm.

[7] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, August 1987.

[8] The http distribution and replication protocol. A WWW Consortium submission, August 1997. http://www.w3.org/TR/NOTE-drp.

[9] R. Farrell. Distributing the web load. *Network World*, pages 57–60, September 22 1997.

[10] Lewis Girod and Karen R. Sollins. Requirements for URN resolution systems. Internet Draft. http://ana-www.lcs.mit.edu/internet-drafts/draft-girod-urn-res-require-00.txt, June 1996.

[11] IBM Interactive Network Dispatcher. http://www.ics.raleigh.ibm.com/netdispatch/.

[12] Large scale network caches provide more bandwidth for your money. Inktomi Corp. White Paper, 1998. http://www.inktomi.com/products/traffic/tech/economics.html.

[13] C. Partridge, T. Mendez, and W. Milliken. RFC 1546: Host anycasting service, November 1993.

[14] Probe Research, Inc. http://www.proberesearch.com.

[15] M. Rabinovich, N. Gehani, and A. Kononov. Scalable update propagation in epidemic replicated databases. *Lecture Notes in Computer Science*, 1057:207–222, 1996. Proc. of EDBT'96.

[16] M. Rabinovich, I. Rabinovich, and R. Rajaraman. Dynamic replication on the Internet. Technical Report HA6177000-980305-01-TM, AT&T Labs, March 1998. Also available as http://www.research.att.com/∼misha/radar/tm.ps.gz.

[17] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[18] D. Terry, A. Demers, K. Peterson, M. Spreitzer, M, Theimer, and B. Welch. Session guarantees for weakly consistent replicated data. In *Int. Conf. on Parallel and Distributed Information Systems*, 1994.

[19] WindDance Networks Corp. Webchallenger. http://www.winddancenet.com/webchallenger/products/frame_products.htm.

[20] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems (TODS)*, 22(4):255–314, June 1997.

[21] Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, and David Culler. Using smart clients to build scalable services. In *1997 Annual Technical Conference, January 6–10, 1997. Anaheim, CA*, pages 105–117. USENIX, January 1997.

Consensus-Based Management of Distributed and Replicated Data

Michel RAYNAL
IRISA, Campus de Beaulieu
35 042 Rennes-cedex, France
raynal@irisa.fr

Abstract

Atomic Broadcast and *Atomic Commitment* are fundamental problems that have to be solved when managing distributed/replicated data. This short note aims at showing that solutions to these problems can benefit from results associated with the *Consensus* problem. Such an approach helps gain a better insight into distributed/replicated data management problems.

More precisely, this note addresses one of the most important issues one is faced to when designing distributed/replicated data management protocols, namely, their *Non-Blocking* property. This property stipulates that the crash of nodes participating in a protocol must not prevent the non-crashed nodes from terminating the protocol execution. Results from the Consensus study allow to know the *minimal assumptions* a system must satisfy in order its distributed/replicated data management protocols be non-blocking despite process crash and asynchrony.
Keywords: Asynchronous Distributed Systems, Atomic Broadcast, Atomic Commitment, Consensus, Crash/no Recovery, Crash/Recovery, Data Management, Non-Blocking, Replicated Data, Transaction.

Asynchronous Distributed Systems

Systems that span large geographical areas (*e.g.*, through Internet), or systems that are subject to unpredictable loads are fundamentally *asynchronous*. This means that it is not possible to assume and to rely on upper bounds for message transfer delays or for process scheduling delays when one has to implement an application (*e.g.*, banking, booking-reservations or point-of-sale commerce) on such a system. This does not mean that timeout mechanisms do not have to be used. It only means that any value used to set a timer can not be trusted when this value is given a system-wide meaning. More precisely, as values used by timers do not define correct upper bounds, the taking of a critical decision by the system or by the upper layer application can not be based on them. That is why asynchronous distributed systems are sometimes called *time-free* systems.

When the system, although asynchronous, suffers no failure (an unrealistic case in practice!) it is possible to compute a consistent snapshot of its state from which consistent decisions can be taken. *Consistent* means here that the snapshot is a past global state the system has or could have passed through. This uncertainty (has/could have) is intrinsically due to the asynchrony of the system.

When failures can occur, the situation becomes much more complicated. The combination of failures (whose occurrences are unpredictable!) and asynchrony create a stronger uncertainty on the system state. Consequently,

Copyright 1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

this can make very difficult or even impossible for an observer (*e.g.*, a process of the application layer) to determine a consistent global state. To address this problem theoreticians have introduced an *abstract* problem, namely the *Consensus* problem, which can be instantiated to solve practical problems dealing with globally consistent decision takings. Among those problems, the *Atomic Broadcast* (AB) problem and the *Non-Blocking Atomic Commitment* (NBAC) problem are particularly important in the distributed/replicated data management context.

This note briefly discusses the connections of the AB and NBAC problems with the *Consensus* problem. More precisely, the *Consensus* problem is used to throw light on these data management problems. It appears that the knowledge of both practical and theoretical results associated with *Consensus* provides basic principles that can help better understand not only solutions to distributed/replicated data management problems but also their intrinsic difficulty. One of the main issues encountered when designing a protocol (that solves such a data management problem in a failure-prone asynchronous system) concerns its *Non-Blocking* property. this property states that, despite the actual but unpredictable pattern of failure occurrences, the protocol must terminate at all non-crashed nodes. The focus of this note is on this *Non-Blocking* property. When a protocol works (correctly terminates), we must understand *why* it does work (correctly terminate). When a protocol does not work, we must understand *why* it does not work. *Knowing the assumptions that are required for data management protocols to correctly terminate in failure-prone asynchronous distributed systems is a fundamental issue.* These assumptions actually define the "limits" beyond which a data management problem may not be solved. Such a knowledge should help researchers and engineers (1) to get a deeper insight into the intrinsic difficulty one has to master when solving distributed/replicated data management problems, and (2) to state the precise assumptions under which their particular distributed/replicated data management problem can be solved.

What is the Consensus Problem?

The Problem

In the *Consensus* problem, each process p_i proposes a value v_i and all non-crashed processes have to decide on some value v that is related to the set of proposed values [10]. Actually, v_i is the *view* the process p_i has of (a relevant[1] part of) the system state. The consensus imposes one of these views to processes. Its aim is, despite failures and asynchrony, to provide processes with the *same view* of (the relevant part of) the system state. After a consensus execution, as processes have the same view of the system state, they can proceed in a globally consistent way (*e.g.*, by taking identical decisions).

Formally, the *Consensus* problem is defined in terms of two primitives: propose and decide. When a process p_i invokes propose(v_i), where v_i is its proposal to the consensus, we say that p "proposes" v_i. In the same way, when p invokes decide and gets v as a result, we say that p "decides" v. It is assumed that links are reliable and that processes can crash. Moreover, a crashed process does not recover. This is the *Crash/no Recovery* model. In this model, a correct process is a process that does not crash[2]. (Lossy links and process recovery are addressed below.) The semantics of propose and decide is defined by the following properties:

- C-Termination. *Every correct process eventually decides.*
- C-Agreement. *No two processes (correct or not) decide differently.*
- C-Validity. *If a process decides v, then v was proposed by some process.*

While C-Termination defines the liveness property associated with the *Consensus* problem, C-Agreement and C-Validity define its safety property.

[1] The word "relevant" has to be appropriately instantiated for each particular problem.

[2] Practically, this means that the process does not crash during the execution of the consensus protocol, or during the execution of the upper layer application that uses the consensus protocol.

About Failures

What is surprising with this apparently simple problem is that it has no solution when processes can crash. This impossibility result is from Fischer, Lynch and Paterson [10] who have formally shown that the *Consensus* problem has no deterministic solution in asynchronous distributed systems that are subject to even a single process crash failure. Intuitively, this negative result is due to the impossibility to safely distinguish (in an asynchronous setting) a crashed process from a slow process, or from a process with which communications are very slow.

This impossibility result has motivated researchers to find a set of minimal assumptions that, when satisfied by a distributed system, makes the Consensus problem solvable in this system. Chandra-Toueg's *Unreliable Failure Detector* concept constitutes an answer to this challenge [7]. From a practical point of view, an unreliable failure detector can be seen as a set of oracles: each oracle is attached to a process and provides it with a list of processes it suspects to have crashed. An oracle can make mistakes by not suspecting a crashed process or by suspecting a non-crashed one. By restricting the domain of mistakes they can make, several classes of failure detectors can be defined. From a formal point of view, a failure detector class is defined by two properties: a *Completeness* property which addresses detection of actual failures, and an *Accuracy* property which restricts the mistakes a failure detector can make.

Among the classes of failure detectors defined by Chandra and Toueg, the class $\diamond\mathcal{S}$ is characterized by the two following properties. The *Strong Completeness* property states: *Eventually, every crashed process is permanently suspected by every correct process.* The *Eventual Weak Accuracy* property states: *There is a time after which some correct process is never suspected.* It has been shown in [8] that, provided a majority of processes are correct, these conditions are the *weakest* ones that allow to solve the Consensus problem. This means that the Consensus problem can not be solved in runs that do not satisfy these two properties. Furthermore, while the completeness property can always be realized by using timeouts, it is important to note that the Eventual Weak Accuracy property can only be approximated in asynchronous distributed systems[3]. So, when the Eventual Weak Accuracy property can not be guaranteed, it is possible that the Consensus problem can not be solved. This means that the Eventual Weak Accuracy property defines very precisely the borderline beyond which the consensus problem can not be solved. This is a very important result that can help understand why some protocols (which can be shown to be instantiations of consensus protocols) work (do terminate) in some circumstances and do not work (can not terminate) in other circumstances.

It is important to note that several consensus protocols based on unreliable failure detectors of the class $\diamond\mathcal{S}$ have been proposed [7, 19, 25]. They all are based on the rotating coordinator paradigm and proceed in consecutive asynchronous rounds. Each round, coordinated by a predetermined process, implements a majority vote strategy. During a round, the associated coordinator tries to impose a value as the decision value. If the Eventual Weak Accuracy property is satisfied, there will be a round during which the associated coordinator will not be erroneously suspected, and this coordinator will succeed in establishing a decision value.

Current Research Efforts

(1) Tradeoff Safety *vs* Liveness. The crucial problem when implementing a consensus protocol lies in the fact its liveness can not be guaranteed in time-free asynchronous unreliable systems. All the previous protocols always guarantee safety, but their liveness can not be taken for granted. When the Eventual Weak Accuracy property of the underlying failure detector is not satisfied, they may not terminate. In that case the protocol does not satisfy the C-Termination property. More precisely, any protocol that requires a failure detector of the class $\diamond\mathcal{S}$ may lose its liveness (*i.e.*, it may not terminate), but will never lose its safety (*i.e.*, if it gives a result, this result is correct) when the underlying failure detector malfunctions by failing to meet its properties. A current research effort concerns the design of consensus protocols that trade liveness against safety. In that case, the resulting protocol always terminates, but there is a price that has to be paid: the safety property is ensured only with some probability; this probability depends on the density function associated with the random variable representing

[3]Otherwise, this would contradict the Fisher-Lynch-Paterson's impossibility result!

message transfer delays [6].
(2) Towards More Realistic models. The previous results have been established in a relatively simplistic model, namely, the *Crash/no Recovery* model. This model assumes that communication is reliable and that a process that crashes during the protocol execution does no recover.

Research is now focusing on more realistic models in which messages can be lost and processes can recover during the protocol execution. This is the *Crash/Recovery* model. Current results concerning this realistic model are : (1) Methods and principles to cope with message loss [2, 4, 9, 15, 20]; (2) Definitions of appropriate failure detectors [2, 9, 20]; (3) Consensus protocols [2, 20]; (4) The statement of a necessary and sufficient condition on the use by each process of a local stable storage in which it can save critical state data during its "crash" periods [2].

The Non-Blocking Atomic Commitment Problem

The Problem

The *Non-Blocking Atomic Commitment* problem (NBAC) has originated from databases, more precisely from transactions. In a distributed system, a transaction usually involves several participant sites (*i.e.*, several processes). At the end of a transaction, its participants are required to enter a commitment protocol in order to commit it (when enough things went well) or to abort it (when too many things went wrong). Each participant votes YES or NO. If for any reason (deadlock, storage problem, concurrency control conflict, etc.) a participant can not locally commit the transaction, it votes NO. Otherwise a vote YES means that the participant commits locally to make updates permanent if it is required to do so. Then, the decision to commit or to abort is determined. The decision must be COMMIT if enough participants (usually all) voted YES. It must be ABORT if too many participants (usually one) voted NO [3, 5, 12].

More formally, NBAC in an asynchronous distributed system, can be defined by the following properties, where the parameter x is used to capture several instances of the problem within a single definition ($x = n$ characterizes the classical *All-or-none* NBAC problem, while $x = \lceil (n+1)/2 \rceil$ characterizes the *Majority* NBAC problem, where n is the total number of participants):

- NBAC-Termination. *Every correct participant eventually decides.*
- NBAC-Agreement. *No two participants decide differently.*
- NBAC-Validity. This property gives its meaning to the decided value. It is composed of three parts.
  - Decision Domain. *The decision value is* COMMIT *or* ABORT.
  - Justification. *If a participant decides* COMMIT, *then at least x participants have voted* YES.
  - Obligation. *If x participants vote* YES *and are not suspected (to have crashed), then the decision value must be* COMMIT.

The justification property states that the "positive" outcome, namely COMMIT, has to be justified: if the result is COMMIT, it is because, for sure, things went well (*i.e.*, enough processes voted YES). Finally, the obligation property eliminates the trivial solution where the decision value would be ABORT even when the situation is good enough to commit.

NBAC **is a Consensus Problem**

Actually the NBAC is a particular instance of the Consensus problem. We provide here a simple protocol that reduces NBAC to Consensus. Figure 1 describes this reduction when we consider an All-or-none NBAC problem (*i.e.*, x is equal to the number of participants).

```
(1)    ∀ q ∈ participants do send(vote) end do;
(2.1)  wait (  (delivery of a vote NO from a participant)
(2.2)        or (∃ q ∈ participants:  p suspects q to have crashed)
(2.3)        or (from each q ∈ participants:  delivery of a vote YES)
(2.4)        );
(3.1)  case
(3.2)        a vote NO has been delivered        → view := ABORT
(3.3)        a participant has been suspected    → view := ABORT
(3.4)        all votes are YES                   → view := COMMIT
(3.5)  end case;
(4)    propose(view); decision:=decide; % Consensus execution %
```

Figure 1: From Consensus to NBAC in Asynchronous Systems

The behavior of every participant p is made of 4 steps. First (line 1), p disseminates its vote to all participants. Then (lines 2.*), p waits until either it has received a NO vote (line 2.1), or it has received a YES vote from each participant (line 2.3), or it has suspected a participant to have crashed (line 2.2). Then (lines 3.*), the participant p builds its own $view$ of the global state: this view is COMMIT if from its point of view everything went well (line 3.4), and ABORT if from its point of view something went wrong (lines 3.2 and 3.3). Finally (line 4), p participates in a consensus. After having proposed its local view of the global state (invocation of propose($view$)), p waits for the result of the consensus execution (invocation of *decide*) and saves it in the local variable $decision$. It can be easily shown that this reduction protocol satisfies the NBAC-Termination, the NBAC-Agreement and the NBAC-Validity properties.

This reduction shows that, in asynchronous distributed systems, the NBAC problem[4] can be solved each time the Consensus problem can be solved. Actually, they are equivalent [13]. It is important to note that the NBAC protocols described in [27] assume a synchronous distributed system: they rely on timeouts to detect participant crashes. So, due to these timeouts, their termination is always guaranteed. But, if the values used to set timers reveal to be erroneous (this occurs when they are not correct upper bounds), those protocols can not satisfy their safety property. In time-free asynchronous systems, the satisfaction of the termination property of NBAC protocols relies on the fact the underlying failure detectors satisfy the Strong Completeness property and the Eventual Weak Accuracy property. Basing these protocols on a solution to the Consensus problem help understand the minimal assumptions that have to be satisfied in order protocols solving NBAC work. More information on the relations between the NBAC problem and the Consensus problem can be found in [13, 17, 23].

The Case of Replicated Data

The replication of a data does not change the problem. In that case, the NBAC problem usually requires that a *majority* of the replicas agree to commit updates. The appropriate version of the NBAC problem is the *Majority* version [5]. The reduction of the *Majority* NBAC problem to the Consensus problem requires to add an additional step to the protocol described in Figure 1. Such reduction protocols are described in [14, 16]. This shows that, although not often claimed in a clear way, the NBAC problem (be it used for committing updates on distinct data

[4]Note that the definition we have considered for the NBAC problem includes the "suspicion of a participant" notion (see the Obligation property) which is less precise than the "crash of a participant" notion: due to asynchrony, a participant q can be suspected by another participant p to have crashed while actually q has not crashed. This shows that, in an asynchronous system, the notion of "suspicion of a participant" is the best approximation we can have for "crash of a participant".

or on copies of a replicated data) is an instantiation of the Consensus problem. This is particularly important as it allows us to know the exact conditions under which a NBAC protocol is non-blocking.

The Atomic Broadcast/Multicast Problem

An increasing number of works consider an *Atomic Broadcast* (AB) primitive to disseminate updates to copies of replicated data [1, 21, 22, 29]. A suite of broadcast primitives has been formally defined in [18].

The Atomic Broadcast Problem

The AB problem is defined by two primitives, namely A_Broadcast and A_Deliver. Informally, the effect of these primitives is to guarantee that processes are delivered messages in the same order. Processes have to agree not only on the set of messages they are delivered but also on their delivery order. More precisely, the AB problem is defined by the three following properties:

- AB-Termination. This property is composed of two parts.
  - Correct Sender. *If a correct process executes* A_Broadcast(m), *then eventually all correct processes execute* A_Deliver(m).
  - Delivery Occurrence. *If a process executes* A_Deliver(m), *then all correct processes execute* A_Deliver(m).
- AB-Agreement. *If two processes* p *and* q *execute* A_Deliver(m_1) *and* A_Deliver(m_2), *then* p *executes* A_Deliver(m_1) *before* A_Deliver(m_2) *if and only if* q *executes* A_Deliver(m_1) *before* A_Deliver(m_2).
- AB-Validity. *If a process executes* A_Deliver(m), *then* m *has been sent by* $sender(m)$ (where $sender(m)$ denotes the sender of m).

As the previous definitions, the definition of what is a "*correct* process" depends on the model we consider. It has been shown that the Consensus problem and the AB problem are equivalent problems: any solution to one of them solves the other [7]. So, any AB protocol designed for asynchronous distributed systems has the same limitations as a Consensus protocol. An AB protocol for Crash/no Recovery systems is described in [7] (here, a correct process is a process that never crash). An AB protocol for Crash/Recovery systems is described in [24] (here, a correct process is a process that can crash and recover an arbitrary number of times, but that eventually recovers and remains "up" long enough for the upper layer application to be able to terminate).

Atomic Multicast

Atomic Multicast to multiple groups (AM) is a primitive particularly useful to manage a set of replicated data. Actually, each data is replicated on a set of sites; these sites define a replication group. The AM primitive allows to reliably send a message to several groups in such a way that (1) all the sites of the destination groups are delivered the message, and (2) there is a single delivery order for all the messages. This means that if 2 copies (of the same data or of different data) receive m_1 and m_2 then they deliver these messages in the same order. When there is a single group, AM confuses with AB. The AM primitive has been investigated in [26] from where the following example is taken.

Let us consider a classical transaction that transfers $1,000 from bank account #1 to bank account #2. To achieve fault-tolerance, assume that each bank account is replicated on several nodes, and assume that every replica is managed by a process. Let g_1 be the fault-tolerant group of processes that manage bank account #1, and let g_2 be the fault-tolerant group of processes that manage bank account #2. The two operations (withdrawal and deposit) can be aggregated into a single message by defining m as: (*remove $1,000 from account #1; add $1,000 to account #2*). When a process in g_1 delivers m, it removes $1,000 from the bank account it manages;

when a process in g_2 delivers m, it adds \$1,000 to the bank account it manages. In this distributed setting, the money transfer transaction can be expressed as the atomic multicast of m to the groups g_1 and g_2. Either both of the groups or none of them deliver m. Moreover, if both groups are destination of the same messages m_1 and m_2, they deliver them in the same order. It is easy to see that the total order property of multicast ensures the serializability of transactions.

An efficient protocol implementing the AM primitive is described in [11]. This protocol is a combination of consensus protocols with a total order multicast protocol designed by Skeen for failure-free systems [28]. What is important is that consensus allows to solve the problem in failure-prone systems. The price that has to be paid is always the same: it is related to the termination property of the AM problem. The knowledge of the Consensus problem indicates to us which are the assumptions related to the detection of failures that have to be satisfied by the underlying system for the AM protocol to terminate.

By way of Conclusion

The aim of this short discussion was to show that the Consensus problem has to be considered as a fundamental problem as soon as one is interested in managing distributed/replicated data despite asynchrony and failure occurrences. In such a context, one of the main problems consists in ensuring the termination (*Non-Blocking*) property of distributed/replicated data management protocols: failed nodes must not prevent non-failed nodes from terminating.

There are two bad news. The first one is that the Consensus problem has no solution in an asynchronous distributed system model as simplistic as the Crash/no Recovery model. This means that it is not possible to design a Consensus protocol that will *always* satisfy the *Non-Blocking* property in presence of process crashes. The second one is that two very important practical problems encountered in the management of distributed/replicated data (NBAC and AB) can be solved in a given system only when the Consensus problem can be solved in this system.

The good news is that a deep knowledge of the Consensus problem (*i.e.*, a knowledge of the assumptions it requires to be solved, and a knowledge of the principles that underlie the protocols that solve it) can provide engineers with concepts and mechanisms that allow them to master problems encountered in the management of distributed/replicated data. As indicated at the beginning of this note, when a protocol works, we must understand *why* it does work. And, when a protocol does not work, we must understand *why* it does not work. Data engineering has to be based on both Technology and Science.

References

[1] Agrawal D., Alonso G., El Abbadi A. and Stanoi I., Exploiting Atomic Broadcast in Replicated databases. *Proc. 1997 EURO-PAR Conference on Parallel Processing*, August 1997, pp. 496-503.

[2] Aguilera M.K., Chen W. and Toueg S., Failure Detection and Consensus in the Crash-Recovery Model. *Proc. of the 12th Int. Symposium on Distributed Computing (DISC'98, ex-WDAG)*, Andros (Greece), Springer Verlag LNCS 1499 (S. Kutten Ed.), pp. 231-245.

[3] Ö. Babaoğlu and S. Toueg., Non-Blocking Atomic Commitment. *Distributed Systems (Second Edition)*, ACM Press (S. Mullender Ed.), New-York, 1993, pp. 147-166.

[4] Basu A., Charron-Bost B., Toueg S., Simulating Reliable Links with Unreliable Links in the Presence of Process Crashes. *Proc. of the 10th Int. Workshop on Distributed Algorithms (WDAG)*, Springer-Verlag, LNCS 1151 (. Babaoğlu and K. Marzullo Eds), pp. 105-122, Bologna, Italy, October 1996.

[5] Bernstein P.A., Hadzilacos V., Goodman N., *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Massachusetts, 370 pages, 1987.

[6] Bollo R., Le Narzul J.-P., Raynal M. and Tronel F., Probabilistic Analysis of a Group Failure Detection Protocol. *Research Report 1209*, IRISA, Rennes, October 1998.

[7] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(1):225–267, March 1996 (A preliminary version appeared in *Proc. of the 10th ACM Symposium on Principles of Distributed Computing*, pp. 325–340, 1991).

[8] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685–722, July 1996 (A preliminary version appeared in *Proc. of the 11th ACM Symposium on Principles of Distributed Computing*, pp. 147–158, 1992).

[9] Dolev D., Friedman R., Keidar I and Malkhi D., Failure Detectors in Omission Failure Environments. *Short paper in Proc. of the 16th ACM Symposium on Principles of Distributed Computing*, pp. 286, August 1997.

[10] Fischer M.J., Lynch N. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, April 1985.

[11] Fritzke U., Ingels Ph., Mostefaoui A. and Raynal M., Fault-Tolerant Total Order Multicast to Asynchronous Groups. *Proc. 17th IEEE Symposium on Reliable Distributed Systems*, Purdue University (IN), pp.228-234, October 1998.

[12] Gray J.N. and Reuter A., *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1070 pages, 1993.

[13] Guerraoui R., Revisiting the Relationship between Non-Blocking Atomic Commitment and Consensus. *Proc. 9th Int. Workshop on Distributed Algorithms* (WDAG95), Springer-Verlag LNCS 972 (J.M. Hlary and M. Raynal Eds), Sept. 1995, pp. 87-100.

[14] Guerraoui R., Oliveira R. and Schiper A., Atomic Updates of Replicated Data. *Proc. of the 2th European Dependable Computing Conference (EDCC2)*, Springer-Verlag, LNCS 1150, pp. 1365-382, Taormina, Italy, October 1996.

[15] Guerraoui R., Oliveira R. and Schiper A., Stubborn Communication Channels. *Research Report*, Département d'informatique, EPFL, Lausanne, Switzerland, July 1997.

[16] Guerraoui R., Raynal M. and Schiper A., Atomic Commit And Consensus: a Unified View. (In French) *Technique et Science Informatiques*, 17(3):279-298, 1998.

[17] Guerraoui R. and Schiper A., The Decentralized Non-Blocking Atomic Commitment Protocol. *Proc. of the 7th IEEE Symposium on Parallel and Distributed Systems*, San Antonio, TX, 1995, pp. 2-9.

[18] Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems (Second Edition)*, ACM Press (S. Mullender Ed.), New-York, 1993, pp. 97-145.

[19] Hurfin M. and Raynal M., A Simple and Fast Asynchronous Consensus Based on a Weak Failure Detector. *Research Report 1118*, IRISA, Rennes, (July 1997), 19 pages.

[20] Hurfin M., Mostefaoui A. and Raynal M., Consensus in Asynchronous Systems Where Processes Can Crash and Recover. *Proc. 17th IEEE Symposium on Reliable Distributed Systems*, Purdue University (IN), pp. 280-286, October 1998.

[21] Kemme B. and Alonso G., A Suite of Database Replication Protocols Based on Group Communication Primitives. *Proc. 18th Int. IEEE Conference on Distributed Computing Systems*, Amsterdam, May 1988, pp. 156-163.

[22] Pedone F., Guerraoui R. and Schiper A., Exploiting Atomic Broadcast in Replicated databases. *Proc. 4th EURO-PAR Conference on Parallel Processing*, Southampton, September 1998, Springer Verlag LNCS 1470, pp. 513-520.

[23] Raynal M., Non-Blocking Atomic Commitment in Distributed Systems: A Tutorial Based on a Generic Protocol. To appear in *Journal of Computer Systems Science and Engineering*, Vol.14, 1999.

[24] Rodrigues L. and Raynal M., Non-Blocking Atomic Broadcast in Asynchronous Crash-Recovery Distributed Systems. *Research Report*, IRISA, Rennes, October 1998, 16 pages.

[25] Schiper A., Early Consensus in an Asynchronous System with a Weak Failure Detector. *Distributed Computing*, 10:149-157, 1997.

[26] Schiper A. and Raynal M., From Group Communication to Transactions in Distributed Systems. *Communications of the ACM*, 39(4):84–87, April 1996.

[27] Skeen D., Non-Blocking Commit Protocols. *Proc. ACM SIGMOD Int. Conference on Management of Data*, ACM Press, 1981, pp. 133-142.

[28] Skeen D., Unpublished communication. Cited in: Birman K. and Joseph T., Reliable Communication in presence of Failures. *ACM Transactions on Computer Systems*, 5(1):47-76, 1987.

[29] Stanoi I., Agrawal D. and El Abbadi A., Using Broadcast Primitives in Replicated databases. *Proc. 18th IEEE Int. Conference on Distributed Computing Systems (IC DCS'98)*, Amsterdam, May 1988, pp. 148-155.

Replication Strategies for High Availability and Disaster Recovery

Robert Breton
Director
Sybase Replication Server Engineering

1 The Business Problem

Businesses are faced with the critical need to ensure the availability of information and continuous operation of their mission-critical systems in the face of potential failures ranging from hardware losses such as disk crashes and CPU failures to catastrophic losses to their computing facilities and communications networks. For example, the Federal Reserve required banks by 1998 to have appropriate disaster recovery procedures in place that would ensure that business interruptions would not exceed four (4) hours. Most of today's disaster recovery strategies are not well suited to meeting such stringent business requirements without suffering significant loss of business information.

While solutions exist to provide tolerance to component failure, the issue of site loss is often more challenging, with potentially dire consequences due to both business interruption and loss of information. While remote backup storage and disaster recovery sites have traditionally satisfied the requirements for disaster recovery for batch-oriented business systems, they are generally inadequate for protecting the information in on-line transaction processing (OLTP) systems. With the introduction of database replication, continuous duplication of critical OLTP information to off-site backup facilities without the high latency inherent in tape backup approaches is now available. Once established, such an environment can be automated to ensure that information is replicated in a timely manner and the switch to backup systems is accomplished with minimal business interruption.

This paper will look at the evolution of such a replication system, Sybase's Replication Server, in supporting disaster recovery and high availability business requirements.

2 Replication Architectures

While the concept of a single federated database offers the advantage of centralized control, the reliance on a central site often requires compromises in accessibility, performance and local autonomy over the business information as dependency on the network and central site availability and scalability grows. In decentralized businesses, the proliferation of distributed computing platforms often means a lack of consistent, timely consolidated information at the corporate level. While the technology exists to support real-time concurrent updates to multiple systems, the performance impact and availability risks do not often justify its use except in the most critical business transactions.

Copyright 1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

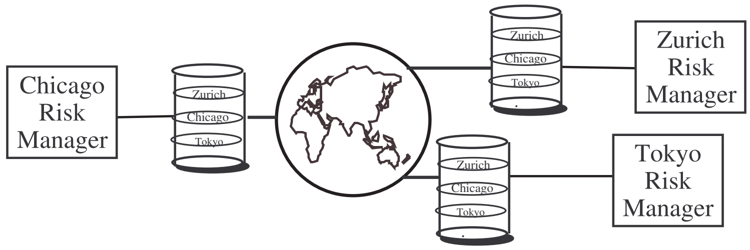

Figure 1: Peer-to-peer replication.

The widespread adoption of clientserver technologies has driven the adoption of more distributed systems architectures. These distributed systems introduce a number of management challenges requiring new strategies for ensuring accurate and timely availability of business information. As the same information is often required in diverse business locations, asynchronous distribution and synchronization satisfies the need to minimize the performance and response time impact of capturing and distributing replicated data while ensuring reliable, timely delivery to replicate sites. This deferred delivery model ensures that the originating application does not wait for the replicated data to be delivered, and any interruption in delivery to one location does not interfere with deliver to other business locations. This provides a high level of information currency, availability and performance, while lowering the cost of delivery. By supporting customization of information as it is delivered, business information delivery can often be tailored to the needs of the each business location.

The following example shows a typical peer-to-peer replication architecture enabling each major business center to manage local information while acting as a backup site to the other locations (see Figure 1). This architecture ensures $7x24$ availability of information throughout the world by eliminating reliance on any single site for information access, while eliminating the dependence on network availability by employing an asynchronous database transaction replication facility.

3 Replication Server: the Publish and Subscribe Model

In 1993, Sybase introduced the first generation of technology providing asynchronous, reliable distribution of database changes. Based on a publish and subscribe model, Replication Server utilizes a highly efficient ”push” architecture, capturing database transactions directly from database recovery logs in near real-time, distributing to subscribing databases through a multi-tiered routing scheme for efficient network utilization, and reliable delivery protocols. Selective distribution using content-based constraints enables subscribers to receive only database changes of interest. By using a ”push” model, latency times between transaction commit at the primary database and delivery to the replicate are minimized. Through intermediate message queue storage, database change messages were logged to ensure reliable delivery and recoverability in the event of interruption in the distribution network. As illustrated in Figure 2 below, database transactions flow through a multi-tiered routing scheme for network efficiency. Each site may select all or a subset of the rows updated at the primary location, based on the content of one or more fields within the row.

4 Warm Standby

The goal of Replication Server is to provide publish and subscribe services for reliable delivery of database transactions. Many customers saw the additional advantages in replication to mirror databases for high availability. Many of the features required to support complex hierarchical distribution and selection publication and subscrip-

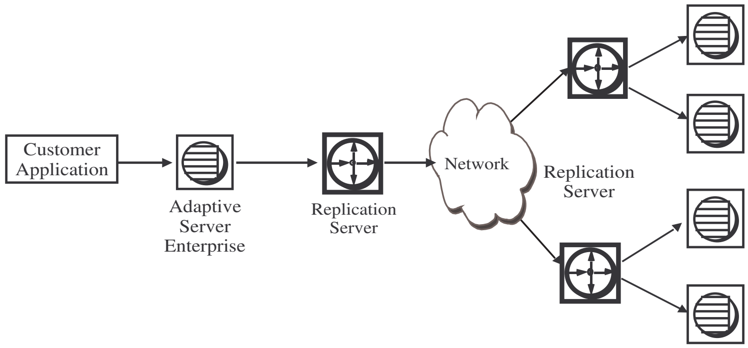

Figure 2: Multi-tiered routing scheme.

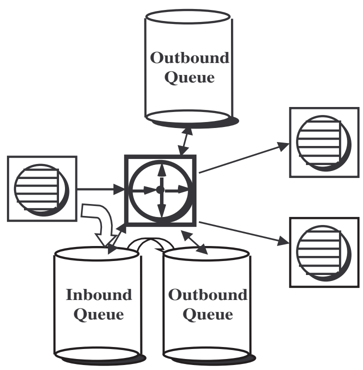

Figure 3: Publish and subscribe model.

tions are less critical in this environment. Because of the simplicity of this requirement, it is possible to simplify internal processing to enable distribution that is more efficient.

In some environments, it is desirable to support this database redundancy for availability combined with standard publish/subscribe services. By creating a special implementation in Replication Server 11 to support this requirement called Warm Standby, administration is simplified as it is no longer necessary to explicit subscribe to replicated tables. Since the Warm Standby implementation appears as a single logical entity to other replication participants, the publish and subscribe environment is not impacted by the additional of Warm Standby.

To contrast publish and subscribe replication vs. warm standby, figure 3 outlines the queues utilized in a standard publish and subscribe model. Transactions from the primary database are first committed to the inbound queue. Inbound queue transactions are filtered based on destinations and subscription criteria, then reordered and delivered to the outbound queues in transaction commit order. Finally, outbound queue transactions are delivered to the replicate databases. In the Warm Standby configuration, there is no need to order transactions or filter subscription criteria. In this model, transactions flow directly from the inbound queue to the standby configuration as shown in figure 4. The administrator marks tables to be replicated in the primary database. Subscriptions are not required to direct transactions to the standby database. The active and standby databases may exchange roles. This is transparent to other publish and subscribe activity against the active/standby database pair.

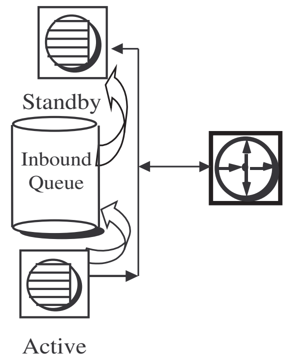

Figure 4: Warm Standby configuration.

5 Schema Replication for Warm Standby

The next evolution of Warm Standby addressed the database administration requirements. Replication Server distributed changes to user-defined tables only. Changes to the structure of tables, security permissions or the maintenance of other objects in the database (stored procedures, rules, triggers, etc.) remained a manual administration process.

In Replication Server 11.5, the concept of Schema Replication was introduced. Schema Replication enables the configuration of Warm Standby to encompass distribution of any operation performed on objects in the database. It also provides a database level configuration command, eliminating the requirement to mark individual tables as replicated objects. Any new objects created are immediately replicated to the standby site. By mirroring all database operations, the administration requirements for Warm Standby are drastically reduced.

6 High Availability Alternatives

There are a number of alternatives for high availability database architectures. This section outlines the major alternatives and the risks and benefits of each approach.

6.1 Hardware-based High Availability Solutions

Hardware and software solutions are available on most platforms to support high availability in a clustered environment. High availability clusters allow multiple CPUs to share common resources, such as disk drives, and provide automatic fail-over in the event of a CPU failure. Cluster solutions generally have severe distance limitations that preclude separating the hardware components adequately to mitigate risks associated with geographic proximity. These include natural disasters, such as fire, floods, tornadoes and earthquakes, and related services outages such as electricity and telecommunications networks.

Distributed disk arrays are emerging that mitigate the site protection issues by providing asynchronous mirroring to physically separate devices for better site protection. While synchronous mirroring satisfy the requirement for committed transactions to be preserved upon a disk or CPU failure, they do present various limitations. Asynchronous mirroring, while extending protection from many site-related failures, has additional risks associated with the potential loss of data. A discussion of benefits and limitations of both synchronous and asynchronous mirroring are presented in Table 1.

Benefits	Limitations
Synchronous disk mirroring with clusters: No loss of committed database transactions on a CPU failure.	Cluster fail-over may requires a significant down-time to perform appropriate restart and recover operations.
Applications generally do not require awareness of physical resource changes, such as network addresses.	Synchronous remote disk mirroring may impact application performance due to network overhead to remote disks.
Asynchronous disk mirroring can provide physical protection by supporting extended physical distances.	Most database systems require special handling to properly recover potential I/O loss in Asynchronous distributions.
Clustered CPU's may be used for other activities when not providing backup services.	Clustered CPU's are not available for use against the same database image.
	No protection from data corruption introduced by the hardware/software.

Table 1: Clusters with high availability disks (RAID/mirroring)

6.2 Software High Availability Solutions

Software high availability solutions are typically support physically separate hardware to provide protection against site loss. For example, a simple implementation of Cold Standby can be accomplished by periodically restoring database backup images and rolling transaction logs to a standby site.

For Hot Standby, applications could update multiple systems simultaneously (synchronous updates), using two-phase commit (2PC) protocols to guarantee transaction integrity. The risk of this type of solution is the potential business interruption if there is a failure in a participating system. Generally, 2PC applications are appropriate for very high-value transactions where consistency across multiple systems is more critical than availability.

Warm Standby offers the protection afforded by redundancy, without the availability constraints of synchronous updates or the time delays of batch-oriented cold backup methods. By providing asynchronous reliable delivery, applications are minimally impacted by the operation of the Warm Standby software system and the availability of the standby system. Benefits and limitations of software-level Warm Standby such as Sybase's Replication Server are presented in Table 2.

7 Warm Standby Considerations

Since a two-phase commit protocol is not being used to synchronize the standby site with the active site, there is a possibility that a failed active site has committed a transaction, but the standby site has not received that transaction. The major advantage replication offers over cold standby methods (such as tape backups) is that transaction loss due to latency is minimized due to the continuous "push" architecture.

New procedures may be required to support the switch over to the standby system, since applications may not be transparently connected to the standby server when a failure occurs. The application can be designed to re-establish its connection to the appropriate database, or switching middleware may be imposed to support this function. It makes economic sense to maximize use of the Warm Standby database during non-failover periods for other business purposes. Warm Standby offers that flexibility as it maintains an online database system. Using the standby site for applications that generate reports, ad-hoc queries, or perform decision support functions is ideal. These applications would normally affect the performance of the active site so this offers the potential to reduce primary site overhead, with some potential impact to the standby. Applications running against the

Benefits	**Limitations**
Ability to quickly swap to the standby system in the event of failure, as standby database is online.	Warm standby system may be ”out-of-date” by transactions committed at the active database that have had sufficient time to reach the standby database.
Warm standby systems can be configured over a Wide Area Network, providing protection from site failures.	Client applications may need to be aware of a swap to the Warm Standby if the active system fails.
Data corruption is not replicated as transactions are logically reproduced rather than I/O blocks mirrored.	Logical transaction reproduction incurs more overhead than I/O block copying.
Originating applications are minimally impacted as replication takes place asynchronously after commit of the originating transaction.	Some overhead is introduced in the application server for transaction capture and delivery to the replication system.
The warm standby database is available for read-only operations such as DSS systems, improving utilization of standby databases.	Network bandwidth, hardware resource requirements and software overhead may limit scalability of software based solutions.

Table 2: Software-level Warm Standby systems.

standby site do need to be cognizant that the data may not be current, but decision-support applications often tolerate this latency.

In a peak volume OLTP environment, the effects of the Warm Standby database falling farther behind the active database's transaction stream must be considered. If the standby site must be operational within five minutes, and the standby system falls 30 minutes behind, then replication may not be the optimal solution. If the standby site is expected to fall 30 minutes behind during peak operations, yet the standby site has one (1) hour to recover, then using Replication Server to maintain the standby site is feasible. Capacity planning is an important consideration in the architecture of a standby system to ensure that not only average performance loads can be achieved, but that peak loads and typical down-time situations can be adequately recovered.

8 Summary

Warm Standby with Replication Server is a significant tool in providing distributed high availability services, affording protection against site failures through asynchronous, wide-area delivery of database transactions. By enabling multiple usage of standby systems, replication provides a cost-effective alternative to redundant hardware that is only utilized for recovery operations. In compliment with traditional hardware high availability solutions, Replication Server extends support for disaster recovery requirements beyond component failure recoverability to significantly improve site protection and reduce potential information loss of cold standby solutions.

CALL FOR PAPERS

The 16th International Conference on

Data Engineering

Tentative Date: Feb. 28 - Mar. 3, 2000
San Diego, CA, USA

Sponsored by
IEEE Computer Society TC on Data Engineering

SCOPE

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The 16th International Conference on Data Engineering will continue in its tradition of being a premier forum for presentation of research results and advanced data-intensive applications and discussion of issues on data and knowledge engineering. The mission of the conference is to share research solutions to problems of today's information society and to identify new issues and directions for future research and development work.

TOPICS OF INTEREST

These include (but are not restricted to):

- Advanced Query Processing
- Data Mining & Knowledge Discovery
- Engine Technology (Storage Management, Access Structures, Recovery, etc.)
- Multimedia & Digital Libraries
- New Departures (Mobile Agents, Embedded Services, Personalization, etc.)
- New Forms of Data (Spatial, Temporal, etc.)
- OLAP & Data Warehouses
- Replication, Availability, & Consistency
- System Administration, Ease of Use, and DB Design
- Web, Distribution, & Interoperation
- Workflow, Transactions, & E-Commerce
- XML & Metadata

IMPORTANT DATES

Abstract submissions (electronic in Ascii): June 9, 1999

Paper submissions (hardcopy or electronic): June 16, 1999

Web page has electronic submission details

Panel/tutorial/industrial submissions: June 16, 1999

Acceptance notification: October 15, 1999

Camera-ready copies: December 1, 1999

ORGANIZING COMMITTEE

General Chair:	P.-Å. (Paul) Larson, Microsoft, USA
Program Co-chairs:	David Lomet, Microsoft, USA Gerhard Weikum, Univ of Saarland, Germany
Panel Program Chair:	Mike Carey, IBM Almaden, USA
Tutorial Program Chair:	Praveen Seshadri, Cornell Univ, USA
Industrial Program Co-Chairs:	Anil Nori, Oracle, USA Pamela Drew, Boeing, USA

PROGRAM VICE-CHAIRS

Advanced Query Processing
Jeff Naughton, Univ of Wisconsin, USA

Data Mining & Knowledge Discovery
Sunita Sarawagi, IBM Almaden, USA

Engine Technology
Hank Korth, Lucent - Bell Labs, USA

System Admin, Ease of Use, & DB Design
Arnie Rosenthal, Mitre, USA

OLAP & Data Warehouses
Jeff Ullman, Stanford Univ, USA

Workflow, Transactions, & E-commerce
Hans Schek, ETH Zurich, Switzerland

XML and Metadata
Phil Bernstein, Microsoft, USA

Web, Distribution, & Interoperation
Donald Kossmann, Univ of Passau, Germany

Multimedia & Digital Libraries
Stavros Christodoulakis, Univ of Crete, Greece

Replication, Availability, & Consistency
Theo Haerder, Univ of Kaiserslautern, Germany

New Forms of Data
Beng Chin Ooi, National Univ of Singapore, Singapore

New Departures (Agents, Mobile, etc.)
H. V. Jagadish, U Illinois - Urbana-Campaign, USA

SUBMISSIONS

Paper, panel, tutorial, or industrial-program submissions must be received by June 16, 1999, 12:00 PM Pacific Time. Paper length must not exceed 20 pages in not smaller than 11 pt font.

For hardcopy submission, seven (7) copies should be sent to:

David Lomet
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399, USA
E-mail: lomet@microsoft.com

For electronic submission, please consult the conference web site at: **http://research.microsoft.com/icde2000**.

An abstract of no more than 300 words in ASCII must be submitted at the conference web site: **http://research.microsoft.com/icde2000** by June 9, 1999. The abstract submission must include the title of the paper, authors' names, an e-mail address of the contact author, a first and second preference among the twelve topical areas, and whether the submission is for the research paper, panel, tutorial, or industrial program.

PANELS AND TUTORIALS

The research and industrial track will be complemented by panels and tutorials. Proposals for each program should be sent to the same address for paper submissions (see there) by **June 16, 1999**.

INDUSTRIAL PROGRAM

The conference program will include a number of papers devoted to industrial developments, applications, and experience in using databases. Papers intended for this program should be clearly marked as industrial track papers at the time of submission.

Conference web site: http://research.microsoft.com/icde2000

Call for Papers
User Interfaces to Data Intensive Systems
UIDIS
Edinburgh 5th - 6th September 1999
http://img.cs.man.ac.uk/UIDIS99

Background

Research into user interfaces to databases and other information management systems is often considered to lag behind research in the underlying technologies. However, in an age where information that is stored anywhere is routinely available anywhere, the need for effective user interfaces to data intensive systems is certainly as great as ever.

This workshop, which builds upon the series of workshops in Interfaces to Database Systems (IDS) held in 1992, 1994 and 1996, seeks to bring together researchers in information management systems and human-computer interaction to exchange ideas and results on how user interfaces to data intensive systems can be made easier to both construct and use. The two day workshop will be held in Edinburgh, the historic capital city of Scotland, between the Interact and VLDB conferences, thereby allowing attendees at either of these major international conferences to attend the workshop, and vice versa. UIDIS will be a limited numbers workshop, to encourage an informal atmosphere and to provide plenty scope for discussion and debate as well as presentations and demonstrations.

Topics of Interest

UIDIS is intended to be an interdisciplinary forum, and invites papers on all aspects of user interfaces to data intensive systems. The following list should thus be taken as indicating topics of particular interest, rather than as a filtering mechanism:

data mining tools	interface architectures	user-interface development systems
distributed information systems	semantic hypermedia	user studies
information retrieval	model-based interfaces	virtual reality
information visualisation	multimedia interfaces	visual information management
intelligent user interfaces	natural language interfaces	visual languages
interactive system design	query interfaces	

Also invited are application papers that can report on experience in medical, scientific, spatial or other challenging domains, and interfaces to different forms of data-intensive systems, such as databases, design tools, digital libraries, etc.

Submission Guidelines

Papers should be up to 5000 words in length, and should be submitted to the address given below. Demonstration proposals should be marked as such, and should be submitted as short papers of up to 1000 words. The proceedings will contain both papers and short reports on demonstrations (it should be assumed that demonstrations will have to be run on stand-alone PCs unless presenters can bring their own hardware).

Important Dates <provisional>

March 1, 1999	Deadline for submission of papers
May 15, 1999	Notification of acceptance/rejection
June 15, 1999	Submission of camera ready copy

Programme Committee

Ghassan Al-Qaimari	Isabel Cruz	Peter Johnson	Claudia Medeiros
Pete Barclay	Alberto Del Bimbo	Jessie Kennedy	Norman Paton
Tiziana Catarci	Max Egenhofer	Wolfgang Klas	Ben Shneiderman
Matthew Chalmers	Carole Goble	Ulrich Lang	Lisa Tweedie
S.K. Chang	Phil Gray	Stefano Levialdi	Dong-kuk Shin
Richard Cooper	George Grinstein	John Mariani	Jean Vanderdonckt
Maria Francesca Costabile			

Contact Details

For further information contact:
Norman Paton
Information Management Group,
Department of Computer Science,
University of Manchester,
Oxford Road, Manchester M13 9PL

OrganisingCommittee
Tony Griffiths
Norman Paton
E-mail: uidis99@cs.man.ac.uk
Tel: +44 (0)161 275 6139
Fax: +44 (0)161 275 6236

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398