A Structural Classification of Integrated Replica Control Mechanisms

Shu-Wie Chen and *Calton Pu*
Department of Computer Science
Columbia University
New York, NY 10027

swfc@cs.columbia.edu
Phone (212) 854-8182
FAX (212) 666-0140

Technical Report CUCS-006-92

Abstract

We classify practical replica control (RC) mechanisms according to two main criteria: (1) the abstract RC protocol, e.g., how many physical copies are written for a logical update, and (2) the kind of integration between the RC mechanism and system services such as atomic broadcast, commit protocols, and concurrency control. We describe about a dozen representative RC mechanisms (most of them implemented) classified by their integration with communications, concurrency control, or both. This classification based on a few essential components of RC mechanisms show that practical replicated systems overwhelmingly adopt the read-one/write-one RC protocol (read/write one physical copy for each logical read/write), due to its low overhead and high availability, despite some inherent inconsistency. Furthermore, this trend points to the need for the integration of weak consistency methods (e.g., epsilon serializability) with read-one/write-one RC protocols if replicated databases are to be realized in practice.

1 Introduction

Replicating objects may improve availability and performance of distributed systems. An object is replicated by storing redundant copies at more than one site. By placing the physical copies at or near sites that often access them, the access time to an object can be reduced significantly, thus increasing performance. By having more than one copy of an object, the object may remain accessible even when a subset of copies is inaccessible, thus increasing availability. Replication of immutable objects is trivial, since copies can be freely reproduced and read. Updating a replicated object introduces the problem of determining the value of an object from the values of its copies. *Replica control (RC)* algorithms are methods that provide such a service.

We classify RC algorithms along two dimensions. First, the abstract RC protocols have some design choices in *how* they map the object into its copies: (1) the number of copies read and written when the object is being read and written, (2) the degree of consistency between read and write operations, and (3) the degree of dynamic reconfiguration. Second, concrete implementations of RC protocols may be integrated with related system services such as communication protocols, distributed agreement protocols, and distributed database concurrency control.

Our contribution is twofold. First, we decompose RC mechanisms into their essential properties and components. Second, our sampling of concrete RC mechanisms shows that in the trade-off between absolute mutual consistency and performance, practical replicated systems invariably choose more performance and availability by tolerating some inconsistency. This paper is not intended to be a comprehensive survey of RC algorithms. Instead, we show some representative examples of each class to illustrate our classification scheme. Furthermore, we compare the structural differences of RC methods but do not analyze their performance due to the structural differences. This is in contrast to cost/benefit analysis of particular RC algorithms such as [11, 32].

To the best of our knowledge, Ceri et al's classification [12] is the only systematic attempt to analyze and compare the structure of RC algorithms. Specifically, they have analyzed RC protocols according to the number of copies read and written, in conjunction with the kind of agreement protocol used in implementation. Our classification may be seen as a refinement of theirs, adding more parameters to the RC protocol dimension such as degree of consistency, and components to the implementation dimension such as concurrency control. In addition, the focus of this paper is on the integration of concrete RC algorithms with system services, an area not covered by Ceri et al's classification.

The paper is organized as follows. Section 2 outlines our structural model and classifies abstract RC protocols. Sections 3, 4, and 5 classify concrete implementations according to the integration of RC protocols with communications services, concurrency control, and both, respectively. The RC methods are compared in Table 1.

2 Replica Control

2.1 Basic Terminology

We model a distributed system as a collection of sites connected by a network. Sites do not share memory or maintain synchronized clocks. They communicate and synchronize only through messages. A replicated system is a distributed system where objects have copies stored redundantly at more than one site. We assume that data structures exist that provide current locations of all copies of each object. We also assume that data manager processes at each site manage the copies stored at the site. The basic operations on objects are *read* and *write*. A read operation returns the value of the object. A write operation stores a value that will be returned by subsequent read operations.

We call a sequence of read and write operations on data objects a *program*. In a non-replicated system, a program's read (write) operation is implemented as a single read (write) on the specified data item. In a replicated system, since there is more than one copy of a data object, a program's read (write) operation may result in a number of reads (writes) on a subset of the copies of a data object. We call the operations on objects at the program level *logical* operations. and the operations on the copies *physical* operations. Similarly, we refer to data objects referenced at the program level as *logical* objects and to copies of the items as *physical* objects.

We denote a logical operation on a logical object X by $R(X)$, $W(X)$, and $O(X)$ for read, write, and read or write, respectively. The corresponding physical operations on the physical copies are denoted by $r(x_i)$, $w(x_i)$, and $o(x_i)$ where x_i denotes the copy stored at site i. As an example, consider an object that is replicated on n sites. A write request of the form $W(X)$ may be translated to the set of physical writes $w(x_1)...w(x_n)$. A program $R(X)W(X)$ may be translated as the set of physical reads $r(x_1)...r(x_n)$ followed by the set of physical writes $w(x_1)...w(x_n)$.

An abstract RC *protocol* is a specification of the mapping from logical operations to physical operations. The specification includes several parameters described in Section 2.2, such as the number and type of the physical operations that must be performed to implement a logical operation and the degree of consistency seen by the user.

An implementation of an RC protocol is an RC *mechanism*, which is a concrete algorithm at the level of messages that implement the RC protocol. The RC mechanisms (described in Sections 3, 4, and 5) may execute on behalf of a logical operation or may perform actions independently of any user program. An example of the former is the execution of physical read operations to obtain a value for a logical read operation. An example of the latter is the propagation of missing updates to a recovering site.

2.2 Main RC Protocol Design Choices

2.2.1 Mapping Logical to Physical

The first important RC protocol design decision is the number of copies that must be physically read (written) when a logical object is being read (written). The three main choices are:

1. Read/Write a (read/write) quorum (usually called *quorum consensus* of size between one and all copies; other copies are ignored.

2. Read One, Write All Available (ROWAA); other copies are handled later.

3. Read One, Write One (ROWO); other copies are handled later.

There are many quorum consensus RC algorithms and we will not list them exhaustively here. They are analogous to hardware fault tolerance techniques such as Triple Modular Redundancy. Gifford's Weighted Voting [20] is a well known early example of these algorithms. In a simplified explanation, quorum consensus RC maps $W(X)$ onto a subset of physical copies, called a write quorum, and $R(X)$ onto an overlapping subset of physical copies, called a read quorum. The existence of the overlap is guaranteed by the pigeonhole principle if the number of copies in the write quorum plus the number of copies in the read quorum exceeds the total number of copies for the object. The overlap itself guarantees that any $R(X)$ will return the result from the last $W(X)$.

The first ROWAA algorithm (Available Copies) was proposed by Bernstein et al [4]. The idea is to write on all copies that are accessible and therefore allow reading of any accessible copy. The copies that are down during an update must be brought up to date, however, during the node recovery time. Compared to quorum consensus, ROWAA offers a faster $R(X)$, just one $r(x)$ instead of several, but requires the knowledge of system configuration.

Some primary copy algorithms (e.g., [3]) are early examples of the ROWO strategy. The basic idea is to allow the secondary copies to lag behind the primary in update, in exchange for faster reading time. Checkpointing schemes such as the Tandem Remote Duplicate Database Facility and backup schemes such as the Remote Backup Procedure [26] are also examples of ROWO. In addition, all of the so-called caching schemes are examples of ROWO, including quasi-copies [2].

These alternatives are not completely mutually exclusive and combinations are possible. For example, the Missing Writes protocol [16] uses ROWAA when all the copies are accessible and switches to quorum consensus when some site or network failures happen.

2.2.2 Degrees of Consistency

The second RC protocol decision is the degree of consistency maintained among replicas. Because of site or network failures, or simple performance considerations, physical copies may contain different values. A fundamental consistency criterion for replicated systems is *mutual consistency*, which requires that the copies of an object agree on a "current" value for the object, in effect, providing the illusion that the copies have the same value [13]. To guarantee mutual consistency, an RC protocol ensures that: (1) physical read operations access appropriate copies to determine the current value (the last value written) of an object, and (2) physical write operations update the appropriate copies so that subsequent physical read operations will return the correct value.

Systems that support mutual consistency provide a property called *one-copy equivalence*: multiple copies of an object appear as a single logical object [1]. One-copy equivalence is a strong consistency property since the replicated system behaves like a non-replicated system but with better availability and performance characteristics. Providing mutual consistency requires a certain degree of cooperation and coordination between the copies of an object. This interaction between sites may restrict the system availability, performance, and autonomy. Therefore, in systems where such factors are critical, it may be desirable to allow some consistency criterion weaker than mutual consistency.

One-copy equivalence is at one end of the spectrum of consistency criteria. We can characterize RC schemes in terms of the time lag (time) or fuzziness (value) of the copies in diverging from one-copy equivalence [41]. As the base case, one-copy equivalence guarantees (1) zero time-lag, that is, it provides "instantaneous" consistency, and (2) a logical read operation will return the value last written. Some systems [41] may only guarantee *eventual* consistency with a time-lag greater than zero. Other proposals such as quasi-copies [2] allow user control of the time lag, imposing constraints on how long replica copies may lag the current object value. Some early systems such as Grapevine [9] place no limit on the time lag and only guarantee that update messages will eventually reach all copies. Some proposals [46] place a restriction on how far a copy's value may diverge from the correct value. These proposals span the spectrum from one-copy equivalence to user-specified degrees of consistency.

Epsilon serializability (ESR) is a family of correctness criteria that explicitly supports such a spectrum of consistency for distributed databases [38, 40]. Many real-world databases have a distance function defined between different states. For example, in a bank account database, the distance between $110 and $80 is $30. This distance function is then used by the user to define how much inconsistency a transaction may tolerate, and the system enforces this limit, allowing Read/Write and Write/Read conflicts to happen without blocking while a transaction has not reached its limit.

Since object consistency and database consistency can be described by a spectrum of values, any system that strives to provide some guarantee for both object and database consistency can also be described by a spectrum. As an example, in a classic replicated distributed database system, both serializability and one-copy equivalence are guaranteed. Replica control and concurrency control are said to provide *one-copy serializability* (1SR): interleaved execution on replicated data is equivalent to serial execution on non-replicated data [13, 5]. We can categorize 1SR systems has handling all types of conflicts, not allowing divergence, and providing instantaneous consistency. We say that such a system provides *strong* consistency since it guarantees the strongest consistency in both dimensions.

2.2.3 Other Properties

Besides the number of copies being read/written and the degree of consistency provided, some other properties further distinguish different RC protocols. For example, many RC protocols assume that physical copies are statically allocated on certain sites. An alternative is to use some kind of copy location protocol (e.g., a directory or broadcast) to dynamically find the copies. This way, physical copies may be regenerated [39] when a site goes down, providing more availability than static RC protocols [32]. Note that regeneration of copies is independent from the issue of dynamic quorum formation in quorum consensus protocols. Regardless of how and when quorum is formed, copy regeneration is possible.

Another possibility is to separate the replication of metadata from the replication of real data. For example, the idea of *witnesses* [33] replicates metadata on the copy (version information) at low cost, due to their small size. The real data (copy itself) can be replicated less often. Some other proposals introduce multiple levels of control into RC. Group paradigm [17] is an example of combining local RC methods with a global RC to provide mutual consistency at the global level.

Ceri et al's classification of RC protocols [12] is the work most closely related to our classification. They have an excellent analysis of abstract RC protocols. In terms of implementation they assume the use of Two-Phase Commit protocol, but do not consider some other important systems services that we describe now.

2.3 Related System Services

The implementation of an RC protocol is non-trivial. Important components include the location of copies, update propagation, and quorum formation and gathering for quorum consensus protocols. These components require system services such as message passing and agreement protocols for atomic update of multiple copies. In a distributed database environment, concurrency control also interacts with the scheduling of physical operations. In this section we discuss how these system services may be used by and integrated into the implementation of an RC protocol.

2.3.1 Communication Mechanisms

Since a replicated system is a distributed system, RC protocols must use some communication mechanism to transmit messages between sites. These messages carry both real data and control information. An RC protocol may adopt synchronous communication such as remote procedure calls, or asynchronous message passing to decrease response time. There are two classes of asynchronous message mechanisms specifically useful to RC protocols.

First, some researchers have proposed some powerful asynchronous group communications primitives that can be called *ordered broadcast*, such as Isis [6], Psync [34], and other schemes described in Section 3. The essential feature of an ordered broadcast is that a message is actually delivered only when all the recipients have agreed on how to place its reception time relative to other messages [24]. The RC protocols may take advantage of this feature to make the ordering on operations at the physical copy level consistent with and according to the ordering of the corresponding operations at the logical object level. For example, logical updates can use ordered broadcast to guarantee mutual consistency without any additional synchronization.

Second, the ROWAA and ROWO protocols need to handle some copies in the background. This can be done with *epidemic propagation* methods including direct mail, anti-entropy, and rumor mongering [14, 21]. Unlike ordered broadcast, epidemic propagation only ensures that the effect of every update will eventually be reflected in all replicas. Direct mail methods transmit update instruction messages directly to the appropriate sites. Though timely and efficient, the method is not reliable because messages may be lost. In anti-entropy methods, a site regularly chooses other sites to exchange messages and resolve differences in their database contents. In rumor mongering, a site sends an update to randomly selected sites until it has tried to share the update with too many sites that have already seen them. Unlike anti-entropy, rumor mongering is not reliable since it is possible for a "rumor" to "die out" before all sites have been informed. A related communication mechanism is *gossip messages* [48]. Like anti-entropy methods, every site eventually learns of each update. However, the gossip algorithm requires that each site maintain information about the states of other sites. Consequently, it can guarantee the causality property that if two updates are causally-related, then if a site knows the second update, then it must also know the first one.

Both the synchronous and asynchronous communication methods can be used for update propagation. However, ordered broadcast primitives are "stronger" than synchronous communication and the epidemic propagation methods because of their implicit synchronization. Therefore, as we shall see in later sections, RC methods that use synchronous request-response methods or epidemic propagation methods will need some additional synchronization, usually in the form of an agreement protocol (e.g., 2 Phase Commit) or concurrency control, respectively, to guarantee mutual consistency.

Agreement protocols are another important building block in distributed computations. In many situations participants need to agree on a value or state, and they use agreement protocols such as the 2 Phase Commit to achieve the consensus. In RC protocols, an agreement protocol is necessary if multiple copies are to be updated consistently. In distributed databases, the outcome of a distributed transaction is decided by an agreement protocol [5]. In a replicated distributed database, RC updates will commit if and only if the transaction commits, so some RC methods integrate their agreement protocol with that of transaction commit protocol, as described in Section 5.

2.3.2 Concurrency Control

Concurrency control methods are algorithms that preserve serializability in databases [5]. Traditional concurrency control methods are conflict-based. Operations conflict if they are from different transactions, they access the same data object, and one of the operations is a write operation. Conflicts are classified as write-write, read-write, and write-read depending on the type and order of the operations. Two transactions are said to conflict if they contain conflicting operations. Concurrency control works by restricting the possible interleavings of the operations of conflicting transactions. More concretely, if a conflict makes two transactions non-serializable, then one of them is either blocked or aborted. In other words, concurrency control is interested in the ordering of operations of conflicting transactions and determines *when* a transaction may perform its read and write operations.

Similarly, RC methods decide on *when* physical operations are carried out given the logical operations. The types of conflicts and their resolution are similar to and somewhat simpler than the problems encountered by concurrency control. However, in a distributed database where transactions may execute concurrently, RC conflicts can occur between operations of the same transaction *and* between operations of different transactions. The interaction between RC operation scheduling and concurrency control operation scheduling can be summarized as follows.

First, an RC method may assume that a concurrency control algorithm will handle the conflicts. This is the case of Gifford's work [20], where each update happens within a transaction. Second, a distributed concurrency control algorithm may assume that each logical operation is atomic. In this case, the same algorithm works for both replicated and non-replicated database, provided that the RC method handles the conflicts between physical operations within a transaction. In both cases, the RC method and concurrency control are scheduling operations at different levels. Some RC methods integrate these scheduling decisions in a way analogous to multi-level concurrency control [45]. They are described in Section 4.

Traditionally, distributed concurrency control must maintain serializability. Consequently, RC methods wishing to allow consistency weaker than serializability cannot be easily integrated. ESR (Section 4.2.1), eventual consistency [41], and other proposals supporting a spectrum of consistency provide new challenges in their integration with appropriate RC protocols.

3 Integration with Communications

We outline in this section some representative RC mechanisms integrated with some kind of communication mechanism. This integration may happen in one of two ways. First, the RC mechanisms may use some form of special broadcast/multicast primitive (Section 3.1). Second, the RC mechanisms may adopt asynchronous update propagation, also called epidemic propagation (Section 3.2). Unless noted, these RC mechanisms are not integrated with distributed concurrency control. Typically, they are used for distributed file systems or name servers. A distributed database is seen as built on top of these RC mechanisms.

3.1 Ordered Broadcast

3.1.1 Virtual Synchrony – ABCAST

The concept of virtual synchrony [8] is based on the premise that event orderings should be subsumed into the communication layer of a distributed system. The atomic broadcast (ABCAST) in the ISIS [7, 8] system sends a message to the recipients atomically, i.e., all at the same time. Some control messages make sure that ABCAST messages have arrived in order at all recipients. This property is useful for RC mechanism in update propagation. An update sent through an ABCAST maps $W(X)$ onto $w(x_i)$ implicitly. As an alternative, causal broadcast (CBCAST) can be used, but since it is integrated with concurrency control, we describe it in Section 5.1.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
None	ROWO	1-Copy Equivalence	Static Mapping	totally-ordered broadcast

Virtual Synchrony with ABCAST

3.1.2 Lazy Replication

Lazy replication [30] is a way to preserve consistency by exploiting the semantics of the service's operations to relax the constraints on ordering. Since some operations need not be ordered identically at all replicas, lazy replication guarantees one-copy equivalence with improved response time and availability. Lazy replication reads and writes on a single copy. Updates are propagated to other replicas by lazy exchange of gossip messages. This replication method uses three types of message ordering: client-ordered, server-ordered, and globally-ordered. Client ordering results in causal or "happens before" order. Server orderings guarantees that operations are performed in the same order at all replicas. Global ordering guarantees that the operation is totally ordered with respect to all other operations. The type of ordering desired depends on the semantics of the operation. One must choose the correct ordering in order to guarantee one-copy equivalence.

The implementation of lazy replication uses ordering information in the form of labels. These labels are generated by the replica services and is used to constrain the execution of operations. In particular, the labels are used to determine whether all updates that an operation depends on are known locally. That replica can either wait for gossip messages from the other replicas or it might send a request to another replica to elicit the information.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
None	ROWO	1-Copy Equiv.	—	client-ordered, server-ordered, etc.

Lazy Replication

3.1.3 Deceit

Deceit is a distributed file system that provides flexible file semantics to allow user control of the degree of efficiency, scalability, and reliability [42, 43]. Deceit requires a program to acquire the write token for a file before updating any copy of the file. The update is then propagated to the other file replicas using ISIS broadcast mechanisms such as CBCAST. Logical read operations, however, can proceed without acquiring any locks and normally access the closest available copy. Similar to Siphon, Deceit's closest copy may be temporarily obsolete. To help users detect the obsolescence, Deceit provides *instability notification*: all available file replicas that may be obsolete because of a concurrent update or an impending propagation are marked as unstable. Any read requests received by unstable replicas are forwarded to the one remaining stable copy. Since the stable copy is the one being updated, reads on this copy will return the most current value. This provides mutual consistency. Serializability is not guaranteed because write tokens are only required to be held over the duration of a write operation, not over the entire program.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
Write Tokens	ROWO	Mutual Consist. (Stability Notif.)	—	epidemic propagation

Deceit distributed file system

3.2 Epidemic Propagation

3.2.1 Grapevine

Grapevine [10] was a distributed system with a replicated registration database. Updates to the registration database use Grapevine's message delivery system, an example of the direct mail asynchronous communication paradigm described earlier. From a user's point of view, an update completes after one copy has been written. Grapevine's message delivery system guarantees that the remaining copies will receive the new update eventually. Unlike ordered broadcasts, there is no synchronization in the reception of the update messages, so different copies of the registration database may be temporarily inconsistent with each other. The applications that use the registration database must be able to tolerate the problem.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
None	ROWO	Temporary Inconsistency	Static Mapping	direct mail

Grapevine registration database

3.2.2 Clearinghouse

Clearinghouse [35] is a production version of Grapevine registration database. It also uses epidemic update propagation. A refinement is that when two programs perform conflicting operations on two different physical copies, Clearinghouse detects the conflicts by comparing timestamps. Upon receiving an update instruction from another data manager, a data manager updates its replica of the object only if update timestamp is later than the timestamp of the previous update to that replica. The update with the most recent timestamp prevails. Read operations may still see temporary inconsistencies due to the asynchronous update propagation.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
None	ROWO	Temp. Incons. Timestamp Confl. Detect.	Static Mapping	direct mail & anti-entropy

Clearinghouse

3.2.3 Global Name Service

The RC scheme developed for a global name service [31] is a further refinement of Clearinghouse. Updates are propagated primarily through direct mail messages. Update operation semantics such as commutativity and idempotency are used to increase performance. However, because direct mail may be unreliable, a sweep-based propagation method is used to guarantee reliable delivery. A sweep is essentially an attempt to bring all copies of an object to the same state by propagating the same set of updates to all copies. However, because the sweep operation is not atomic, additional update operations may occur during the sweep thereby introducing a new inconsistency. Furthermore, since there is no synchronization, no consistency can be guaranteed for operations that are executed after the latest sweep: the result of a read depends solely on which updates have reached the copy being read.

Concurrency	Replica Control			Communications
Control	Mapping	Consistency	Other	Mechanism
None	ROWO	Temp. Incons. Confl. Detect.	Oper. Semantics: Commutat. & Idempot.	direct mail & reliable sweeps

Global Name Service

3.2.4 Oscar

The Open System for Consistency and Replication (OSCAR) is an architecture for weak-consistency replication systems [15]. OSCAR's replication scheme is based on two types of cooperating agents called *replicators* and *mediators*. Replicators perform local physical operations and initiate the propagation of updates to other replicas using unreliable multicasts to implement the direct mail communication method. Mediators guarantee that each replica receives all relevant update instructions by periodically polling replicators to implement the anti-entropy communication method. This hybrid communication scheme ensure reliable, yet timely asynchronous update propagation. Specific forms of weak consistency is achieved by controlling the order and timing of update delivery to a database. Replicators make their decisions to deliver updates independently.

Concurrency	Replica Control			Communications
Control	Mapping	Consistency	Other	Mechanism
None	ROWO	Temporary Inconsistency	Oper. Semantics: Commutat. & Idempot.	direct mail & anti-entropy

OSCAR

3.2.5 Ficus

The Ficus replicated file system provides a replication service on top of the Network File System (NFS) [36, 19]. Ficus uses asynchronous update propagation in the form of asynchronous multicasts, an example of the direct mail communication method. Because the frequency of conflicting updates is low in general-purpose file systems, Ficus adopts a non-serializable correctness policy called *one-copy availability* which allows the update of any copy of an object without requiring a particular copy or a minimum number of copies to be accessible. Ficus also detects conflicting updates using version vectors [18] when they occur, but leaves their resolution to the higher levels in the system.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
None	ROWO	1-Copy Avail. Confl. Detect.	Oper. Semantics Some Confl. Resol.	Asynchronous Multicast

FICUS Distributed File System

3.2.6 Coda

The Coda distributed file system [27, 28] uses two levels of replication to increase availability. *First-class* replicas are traditional replicas that are stored at server sites. *Second-class* replicas are cached copies of first-class replicas that are stored on client machines. Both classes of replicas use an optimistic replica control strategy: read and write operations are allowed everywhere with conflicts detected and handled the operations have been performed. The conflict resolution relies on operation semantics and also manual intervention. The primary concern is to provide continued file availability during disconnected operations.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
None	ROWO	Temp. Incons. Confl. Detect.	Oper. Semantics Some Confl. Resol.	RPC & threads

Coda Distributed File System

4 Integration with Concurrency Control

We outline in this section some RC mechanisms used in distributed databases that are integrated with concurrency control mechanisms. We divide them into two classes, those that support serializability (Section 4.1) and those that support weaker notions of database correctness (Section 4.2). Unless noted, these RC mechanisms are not integrated with specific communications primitives.

4.1 Serializability Based

4.1.1 Piggybacked and Concurrent Updates

The piggybacked and concurrent update techniques [25] are based on the observation that concurrency control (CC) mechanisms already provide synchronization facilities and that RC mechanisms, instead of duplicating these synchronization facilities, could exploit them to improve the combined performance. For example, the delay of waiting for acknowledgements from physical writes to remote copies can be reduced significantly by piggybacking update information on CC synchronization messages. A $W(X)$ is considered complete when $w(x_i)$ have been generated (but not necessarily sent). Since a CC mechanism synchronizes the interaction between conflicting transactions, a transaction's $w(x_i)$ messages do not need to be sent and processed until the next transaction's $r(x_i)$. Since the CC mechanism sends synchronization messages between sites to determine the ordering of $w(x)$ and $r(x)$, $w(x_i)$ messages can be piggybacked on CC control messages and be guaranteed to arrive before the next $r(x_i)$. The receiving sites must process the piggybacked $w(x_i)$ before processing the control messages themselves. This scheme allows asynchronous updates since $w(x_i)$ can be deferred until their results are actually needed by another transaction. Piggybacking update instructions has the additional benefit of reducing message traffic.

Although piggybacked update eliminates the latency for waiting for acknowledgements, it requires that a transaction waiting for a synchronization message to suffer additional delays while the piggybacked update instructions are being processed. This latency can be avoided by decoupling the ordering decision of $w(x_i)$ from the actual processing of $w(x_i)$ themselves. The concurrent update scheme is a modification to the piggybacked scheme that addresses the above problem.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
Confl.-Based e.g., 2PL	Read-One Write-Zero	Serializability & 1-Copy Equiv.	Unspecified	Any, Including ABCAST and CBCAST

Piggybacked and Concurrent Update Scheme

4.1.2 Thomas's Majority Voting

Thomas's majority consensus algorithm [44] is another solution to the update synchronization problem for replicated databases that integrates the RC mechanism with concurrency control. The algorithm strives to preserve both database and object consistency.

To preserve database consistency, a CC scheme comprised of timestamps and intersecting write sites is used to ensure that updates are based on up-to-date information, i.e., the values that the updates depended on have not changed since they were last read. Such CC conflicts are detected at the intersecting write sites by comparing timestamps.

To preserve object consistency, an RC scheme comprised of timestamps is used to ensure that the latest update prevails at each copy. Since updates are not guaranteed to be delivered in the correct order, each site must compare the timestamp of the update instruction with that of the copy. The update is applied only if its timestamp is later than that of the copy. The read access will obtain the latest copy if a sufficiently large read quorum is acquired so it overlaps with the write quorum.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
Confl.-Based Timestamps	Quorum Consensus	Serializability & 1-Copy Equiv.	Static	broadcast or daisy chain

Thomas Majority Voting

4.1.3 Consensus Locking

Consensus locking [23] is a generalization of general quorum consensus [22]. It is a type-specific replication method that relies on an underlying CC scheme to ensure atomicity. As in other quorum consensus schemes, consensus locking requires that a logical operation access a quorum of copies. For correct execution, certain quorums are required to have nonempty intersections. Read quorum intersecting write quorum ensures object consistency.

To maintain database consistency, consensus locking uses strict two-phase locking at each quorum site in conjunction with quorum intersections to detect conflicting operations from different transactions. This is necessary since a conflict can be detected only if the conflicting locks are detected at a site that is in the quorum intersection.

To maintain object consistency, consensus locking uses timestamps in conjunction with quorum intersections. The quorum intersection is needed to guarantee that each value read is the value most recently written. The timestamps are used to order the updates to each copy to ensure that the latest update prevails.

Concurrency	Replica Control			Communications
Control	Mapping	Consistency	Other	Mechanism
Strict 2PL (quorum sites)	Gen.Quo. Consensus	Serializability & Confl. Detect.	Timestamp order. Updates	RPC

Consensus Locking

4.2 Weak Consistency

4.2.1 Epsilon Serializability

Epsilon serializability (ESR) [38, 40] offers a spectrum of consistency criteria spanning serializability on the one end and complete freedom from synchronization on the other end. A distance measure in the database state space allows users to specify the amount of inconsistency tolerated by a transaction and the database system enforces that limit through algorithms called divergence control methods [47]. When applied to replication, ESR allows the replicas to diverge by a limited amount of difference in value, the number of updates, or other distance measures.

Concurrency	Replica Control			Communications
Control	Mapping	Consistency	Other	Mechanism
Divergence Control	ROWO ROWAA	ESR	—	Primarily epidemic propa.

Epsilon Serializability

5 Integration with Concurrency Control and Communications

Some of RC mechanisms attempt to integrate with both CC mechanisms and communication mechanisms. This kind of tight integration tends to lower replication overhead, but also produces less modular systems.

5.1 Virtual Synchrony – CBCAST

In section 3.1.1, we described how the ABCAST totally-ordered broadcast primitive could be used to guarantee mutual consistency in a replicated system that did not provide any CC support. The ABCAST primitive can be used in conjunction with a CC mechanism to guarantee both database and object consistency. However, such a combination is inefficient because services are duplicated. CC guarantees the program exclusive access to an object and consequently exclusive access to all the copies of an object. The ABCAST primitive guarantees that messages sent to copies of the object will be ordered consistently with all other messages, even those that do not affect the object. What is needed is a causal-ordered broadcast primitive that only orders messages sent by transactions that are causally related, i.e., have conflicting operations.

In virtual synchrony, the causal-ordered broadcast primitive is implemented by the ISIS broadcast primitive called CBCAST. By using CBCASTs and some type of CC to acquire exclusive access of objects, a program can be guaranteed to read objects in a mutually-consistent manner. Psync [34] is another example of RC mechanism using causal broadcasts. The difference is that as proposed Psync is not integrated with a CC mechanism.

Concurrency	Replica Control			Communications
Control	Mapping	Consistency	Other	Mechanism
Locking	ROWAA	Serializability & 1-Copy Equiv.	—	CBCAST

Virtual Synchrony with CBCAST

5.2 Siphon

Siphon is a replica management system used to manage multi-site software development efforts [37]. Copies of objects are stored at all sites. Like Grapevine, Siphon allows temporary inconsistencies to arise. Updates are propagated using anti-entropy and may take indefinitely long time to arrive at all sites. However, unlike Grapevine, Siphon prevents simultaneous updates at different sites by requiring programs to acquire write locks before updating an object. The write lock is granted only if the requesting program's local copy contains the current value of the object. This is done by comparing version identifiers stored with each copy. Consequently, a site cannot initiate an update on an object until it has received the latest update on that object from other sites. Read operations always return the local copy without any synchronization.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
Locking	ROWO	Inconsistent Reads	Consistent Updates	anti-entropy

Siphon

5.3 Non-Blocking Quorum Consensus

Nonblocking quorum consensus (NBQC) is an RC scheme that uses an asynchronous propagation mechanism but guarantees correctness by using the synchronization services of CC and atomic agreement protocol [1]. NBQC uses Gifford's weighted voting protocol (Section 2.2) but integrates the implementation of RC with both CC and communications mechanisms. It assumes that the CC protocol will only permit recoverable executions, that is, transactions are only allowed to observe values of objects that are produced by committed transactions. Further, NBQC assumes the deferred update scheme for database recovery, performing the actual physical write operations on copies only when a transaction commits. It also requires a two-phase commit protocol to guarantee the atomicity of transactions that span multiple transactions.

NBQC is similar to the piggybacked update technique in that it uses CC synchronization messages to propagate updates. However, since NBQC uses deferred updates, updates are only propagated at commit time, which coincides with the second phase of the two-phase commit protocol. However, instead of using the standard synchronous implementation of two-phase commit which requires acknowledgements in the second phase, NBQC uses *gossip* messages to asynchronously propagate the commit decision. Database consistency is preserved, however, because the update instructions are piggybacked on the commit decision gossip messages and are processed at each site before commitment.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
Confl.-Based	Quorum Consensus	Serializability 1-Copy Equiv.	Assume DDB	gossip messages

Non-Blocking Quorum Consensus

5.4 N-Ignorance

In an N-ignorant system [29], a transaction may not see the results of at most N prior transactions that it would have seen if the execution had been serial. In such as system, concurrency is increased $(N+1)$-fold with respect to a serializable system at the expense of bounded violations of the database's integrity constraints.

N-ignorance is implemented using quorum locking [23] and gossip messages. However, unlike the original quorum locking, a transaction's read and write quorums are over the same sets (and therefore of the same size) and need not contain a majority of the sites. As a consequence, conflicting transactions may execute concurrently over different replicas of the same object. By adjusting the quorum size and the amount of allowed difference between knowledge states of non-quorum sites and the executing site, an N-ignorant system can control the amount of divergence from serializable executions. Gossip messages are used to broadcast information between sites. They limit the extent to which a site may be ignorant of the effects of prior transactions executed elsewhere. Gossip messages may be piggybacked on CC lock messages or may be sent independently.

Concurrency Control	Replica Control			Communications Mechanism
	Mapping	Consistency	Other	
Confl.-Based	Consensus Locking	Bounded Inconsistency	—	causal broadcast (gossip messages)

N-Ignorance Systems

6 Conclusion

We have proposed a structural classification of replica control (RC) mechanisms. This classification decomposes the RC mechanism into a few essential components. First, the RC protocol specifies how the logical read/write operations are mapped into physical operations on the copies and what degree of consistency is maintained among the copies. For

RC Method	Concurrency Control	Replica Control Mapping	Replica Control Consistency	Replica Control Other	Communic. Mechanism
Virt.Sync.	None	ROWO	1-Copy Equivalence	Static Mapping	ABCAST CBCAST
Lazy.Repl.	None	ROWO	1-Copy Equiv.	—	client-ordered....
Deceit	Write Tokens	ROWO	Mutual Consistency	—	ISIS (CBCAST)
Grapevine Clearinghouse	None	ROWO	Temporary Inconsistency	Static Mapping	direct mail
GNS	None	ROWO	Temp. Incons. Confl. Detect.	Oper. Semantics	dir.mail & rel.sweeps
OSCAR	None	ROWO	Temporary Inconsistency	Oper. Semantics	dir.mail & anti-entropy
FICUS	None	ROWO	1-Copy Avail. Confl. Detect.	Oper. Semantics	Async. Multicast
Coda	None	ROWO	Temp. Incons. Confl. Detect.	Oper. Semantics	direct mail
Piggybacked	Confl.-Based e.g., 2PL	Read-One Write-Zero	Serializability & 1-Copy Equiv.	Unspecified	Any, CBCAST
Thomas Voting	Confl.-Based Timestamps	Quorum Consensus	Serializability & 1-Copy Equiv.	Static	RPC
Consensus Locking	Strict 2PL	Gen.Quo. Consensus	Serializability & Confl. Detect.	Timestamp order	RPC
ESR	Divergence Control	ROWO ROWAA	ESR	—	Primarily epi.propa.
Siphon	Locking	ROWO	Inconsistent Reads	Consistent Updates	anti-entropy
Non-Block.	Confl.-Based	Quorum Consensus	Serializability 1-Copy Equiv.	Assume DDB	gossip messages
N-Ignorance	Confl.-Based	Consensus Locking	Bounded Inconsistency	—	gossip messages

Table 1: Summary and Comparison

example, quorum consensus requires reading and writing of a subset of copies to maintain mutual consistency. In contrast, many distributed file systems require reading and writing of only one copy, allowing temporary inconsistencies among copies. Second, the implementation of an RC mechanism typically is integrated with some systems services, including communications primitives such as causal broadcast and distributed database concurrency control such as locking.

Instead of analyzing abstract replication algorithms, we applied the classification to practical RC mechanisms, about a dozen replicated systems that have been implemented to some degree plus a few theoretical RC algorithms. The comparison of these RC mechanisms on a small number of components and properties is summarized in Table 1. Two observations become apparent from the table.

First, integration with some kind of system service is an important part of practical RC mechanisms. Even if an abstract RC algorithm has been implemented using communications or transaction facilities as building blocks, that implementation would have been considered a prototype or demonstration. A replicated system designed and implemented for practical use invariably integrate some part of the RC mechanism with underlying system services, either communications primitives, commit protocols, or distributed concurrency control.

Second, all the practical RC mechanisms in Table 1 are in the ROWO (read one copy, write one copy) class. An ROWO protocol returns from a logical operations after it has been applied to only one physical copy. This is in sharp contrast to the amount of replication work dedicated to voting and quorum consensus. The few examples in Table 1 that use quorum consensus have not been implemented. Although we have not finished a comprehensive survey, the table shows that in practice RC mechanism designers consider the performance and availability gained by ROWO to be significant enough to offset the inconsistency problems inherent in those systems.

These two observations lead to an important implication in replicated database research. Traditionally, replicated databases have assumed the support for serializability through classic concurrency control algorithms. A natural integration of RC with classic concurrency control yields one-copy serializability, a strong consistency property. Although acceptable in theory, this line of investigation is not compatible with the clear trend in practical RC mechanism work without concurrency control. The trend to weaker consistency properties shows that the challenge in the design of practical replicated databases is in integrating some form of weak consistency criterion with ROWO protocols that offer low overhead and high availability as well as performance. An example of promising approaches in this direction is epsilon serializability (ESR [38], outlined in Section 4.2.1) and divergence control methods [47] that guarantee ESR. Integrating ROWO protocols with ESR and asynchronous communication mechanisms may satisfy the performance, availability, and scalability requirements of practical replicated databases.

References

[1] Divyakant Agrawal and Arthur J. Bernstein. A nonblocking quorum consensus protocol for replicated data. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):171–179, April 1991.

[2] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval systems. *ACM Transactions on Database Systems*, 15(3):359–384, September 1990.

[3] P.A. Alsberg and J.D. Day. A principle for resilient sharing of distributed resources. In *Proceedings of the Second International Conference on Software Engineering*, pages 562–570, 1976.

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

[5] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[6] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the Eleventh Symposium on Operating Systems Principles*, pages 123–138. ACM/SIGOPS, November 1987.

[7] Kenneth P. Birman. Replication and fault-tolerance in the isis system. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 79–86, December 1985.

[8] Kenneth P. Birman and Thomas Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the Eleventh Symposium on Operating Systems Principles*, pages 123–138. ACM/SIGOPS, November 1987.

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

[10] Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

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

[12] Stefano Ceri, Maurice A.W. Houtsma, Arthur M. Keller, and Pierangela Samarati. A classification of update methods for replicated databases. Technical report, Department of Computer Science, Stanford University, 1991.

[13] Susan B. Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.

[14] Alan Demers and et al. Epidemic algorithms for replicated database maintenance. Technical report, Xerox PARC, CSL-89-1, 1991.

[15] Alan R. Downing, Ira B. Greenberg, and Jon M. Peha. Oscar: A system for weak-consistency replication. In Luis-Felipe Cabrera and Jehan-Francois Paris, editors, *Proceedings of the Workshop on Management of Replicated Data*, pages 26–30, Los Alamitos, California, November 8-9 1990. IEEE Computer Society Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society Press.

[16] D.L. Eager and K.C. Sevcik. Achieving robustness in distributed database systems. *ACM Transactions on Database Systems*, 8(3):354–381, September 1983.

[17] A. El Abbadi and S. Toueg. The group paradigm for concurrency control protocol. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):376–386, September 1989.

[18] D. Scott Parker et al. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering*, 9(3):240–247, May 1983.

[19] Richard Guy et al. Implementation of the ficus replicated file service. In *Proceedings of the Summer 1990 USENIX Conference*, pages 63–71, Berkeley, CA, June 11-15 1990. The USENIX Association.

[20] D.K. Gifford. Weighted voting for replicated data. In *Proceedings of the Seventh Symposium on Operating Systems Principles*, pages 150–162. ACM/SIGOPS, December 1979.

[21] Richard A. Golding. Group membership in the epidemic style. Technical report, University of California, Santa Cruz, UCSC-CRL-91-32, 1991.

[22] Maurice Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32–53, February 1986.

[23] Maurice Herlihy. Concurrency versus availability: Atomicity mechanisms for replicated data. *ACM Transactions on Computer Systems*, 5(3):249–274, August 1987.

[24] Thomas A. Joseph and Ken P. Birman. *Distributed Systems*, chapter 14. ACM Press, 1989.

[25] Thomas A. Joseph and Kenneth P. Birman. Low cost management of replicated data in fault-tolerant distributed systems. *ACM Transaction on Computer Systems*, 4(1):54–700, February 1986.

[26] R.P. King, N. Halim, H. Garcia-Molina, and C. Polyzois. Management of a remote backup copy for disaster recovery. *Transactions on Database Systems*, 16(2):338–368, June 1991.

[27] James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[28] James Jay Kistler. Increasing file system availability through second-class replication. In Luis-Felipe Cabrera and Jehan-Francois Paris, editors, *Proceedings of the Workshop on Management of Replicated Data*, pages 65–69, Los Alamitos, California, November 8-9 1990. IEEE Computer Society Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society Press.

[29] Narayanan Krishnakumar and Arthur J. Bernstein. Bounded ignorance in replicated systems. In *Proceedings of the 1991 Principles of Database Systems*, pages 63–74, June 1991.

[30] Rivka Ladin, Barbara Liskov, and Liuba Shrira. Lazy replication: Exploiting the semantics of distributed services. In *Proceedings of the 1990 Symposium on Principles of Distributed Computing*, pages 43–57, 1990.

[31] Butler W. Lampson. Designing a global name service. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–10, 1986.

[32] D.D.E. Long and J.-F. Paris. Regeneration protocols for replicated objects. In *Proceedings of the Fifth International Conference on Data Engineering*, pages 538–545, Los Angeles, 1989.

[33] D.D.E. Long and J.-F. Paris. Voting with regenerable volatile witnesses. Technical Report UCSC-CRL-90-09, Computer and Information Sciences, University of California at Santa Cruz, July 1990.

[34] Shivakant Mishra, Larry L. Peterson, and Richard D. Schlichting. Communication substrate for maintaining replicated data. In Luis-Felipe Cabrera and Jehan-Francois Paris, editors, *Proceedings of the Workshop on Management of Replicated Data*, pages 125–127, Los Alamitos, California, November 8-9 1990. IEEE Computer Society Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society Press.

[35] D.C. Oppen and Y.K. Dalal. The Clearinghouse: A decentralized agent for locating named objects in a distributed environment. *ACM Transactions on Office Information Systems*, 1(3), July 1983.

[36] Gerald J. Popek, Richard G. Guy, Jr. Thomas W. Page, and John S. Heidemann. Replication in ficus distributed file systems. In Luis-Felipe Cabrera and Jehan-Francois Paris, editors, *Proceedings of the Workshop on Management of Replicated Data*, pages 5–10, Los Alamitos, California, November 8-9 1990. IEEE Computer Society Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society Press.

[37] Francis J. Prusker and Edward P. Wobber. The siphon: Managing distant replicated repositories. In Luis-Felipe Cabrera and Jehan-Francois Paris, editors, *Proceedings of the Workshop on Management of Replicated Data*, pages 44–47, Los Alamitos, California, November 8-9 1990. IEEE Computer Society Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society Press.

[38] C. Pu and A. Leff. Replica control in distributed systems: An asynchronous approach. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 377–386, Denver, May 1991.

[39] C. Pu, J.D. Noe, and A. Proudfoot. Regeneration of replicated objects: A technique and its Eden implementation. *IEEE Transactions on Software Engineering*, SE-14(7):936–945, July 1988.

[40] K. Ramamrithan and C. Pu. A formal characterization of epsilon serializability. Technical Report CUCS-044-91, Department of Computer Science, Columbia University, 1991.

[41] Amit Sheth and Marek Rusinkiewicz. Management of interdependent data: Specifying dependency and consistency requirements. In Luis-Felipe Cabrera and Jehan-Francois Paris, editors, *Proceedings of the Workshop on Management of Replicated Data*, pages 133–136, Los Alamitos, California, November 8-9 1990. IEEE Computer Society Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society Press.

[42] Alex Siegel, Kenneth Birman, and Keith Marzullo. Deceit: A flexible distributed file system. In Luis-Felipe Cabrera and Jehan-Francois Paris, editors, *Proceedings of the Workshop on Management of Replicated Data*, pages 15–17, Los Alamitos, California, November 8-9 1990. IEEE Computer Society Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society Press.

[43] Alex Siegel, Kenneth Birman, and Keith Marzullo. Deceit: A flexible distributed file system. In *Proceedings of the Summer 1990 USENIX Conference*, pages 51–62, Berkeley, CA, June 11-15 1990. The USENIX Association.

[44] R.H. Thomas. A majority consensus approach to concurrency control. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

[45] G. Weikum. Principles and realization strategies of multilevel transaction management. *ACM Transactions on Database Systems*, 16(1):132–180, March 1991.

[46] G. Wiederhold and X. Qian. Consistency control of replicated data in federated databases. In *Proceedings of the Workshop on Management of Replicated Data*, pages 130–132, Houston, November 1990.

[47] K.L. Wu, P. S. Yu, and C. Pu. Divergence control for epsilon-serializability. In *Proceedings of Eighth International Conference on Data Engineering*, Phoenix, February 1992. IEEE/Computer Society.

[48] G.T. Wuu and A.J. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the Third ACM Symposium of Principles of Distributed Computing*, pages 233–242, August 1984.