

# Integrating Security with Fault-tolerant Distributed Databases\*

D. AGRAWAL AND A. EL ABBADI

Department of Computer Science, University of California, Santa Barbara, CA 93106, USA

*Replication reduces the security of data in a distributed environment. In this paper, we address the issue of maintaining security in a fault-tolerant replicated database. We present a protocol that combines both security and reliability aspects in a database system. Although this protocol provides the desired level of security, it does so at the expense of availability. By integrating a propagation mechanism with our protocol, we are able to achieve a high level of security and availability.*

Received April 1989.

## 1. INTRODUCTION

In a distributed database system, data is often replicated on several sites to achieve fault-tolerance. Such replication allows the database to remain available to the users in spite of site and communication failures. However, by increasing the availability of data, replication increases the "availability" of the data to intruders or illegal users of the system. Earlier when there was no replication, only one copy of a critical data object existed in the entire network. When we introduce replication, there are several copies of the same data object present in the network, making it more vulnerable. If an intruder manages to access *any* copy of this data object, the security of the data may be compromised. Thus, most fault-tolerant database systems jeopardize the security of data. In this paper we address the issue of maintaining security in a fault-tolerant replicated database.

A distributed system consists of a set of sites connected by a communication network. We assume that both sites and communication links may fail by crashing, and that combinations of such failures may lead to *partitioning failures*.<sup>3</sup> Sites in a partition can communicate with each other, but, not with sites in other partitions. The database consists of a set of objects (or simply files), which are implemented by copies that reside on different sites. Users access the database by issuing transactions that consist of read and write operations. To be consistent, the system should behave as if each object has only one copy in so far as the users can tell. Furthermore, if operations of different transactions are interleaved, the system must behave as if all the transactions are executed in a serial order. These two concepts have been formalized as *one-copy serializability*.<sup>2</sup>

A simple example of a protocol that ensures one-copy serializability, is one that uses two-phase locking<sup>4</sup> to synchronize the interleavings of different transactions, and where a write operation writes all copies of an object while a read operation reads any copy. Although correct, this protocol is not fault-tolerant to the failure of even one site: if one site fails, no write operation may be executed on any object with copies residing on that site. Furthermore, the protocol provides no

security, since if an intruder acquires *one* copy, all the information about the object is compromised.

In order to provide fault-tolerance, Gifford proposed the *quorum protocol*,<sup>6</sup> where associated with each object,  $x$ , is a *read quorum*,  $q_r[x]$ , and a *write quorum*,  $q_w[x]$ . Each copy has associated with it a *version number*, which is initialized to one. A write of object  $x$  is executed by writing  $q_w[x]$  copies of  $x$  and updating their version numbers to be greater than the maximum version number associated with any of those copies. A read of object  $x$  is executed by accessing  $q_r[x]$  copies of  $x$  and reading the value associated with the highest version number. To ensure correctness, any set of size  $q_w[x]$  must have at least one copy in common with any set of size  $q_w[x]$  or  $q_r[x]$ . Since the write quorum can be less than all copies of an object, write operations in this protocol may be executed even when half the copies of an object are inaccessible. However, this protocol does not ensure any level of security of the data, since every copy of an object contains complete information about that object (although this information may not be up-to-date). Hence, an intruder, who can acquire a single copy of an object, jeopardizes the security of the database.

Several schemes have been proposed to achieve security in distributed systems. However, there have been relatively few attempts to integrate security with fault-tolerant databases. Randell and Dobson<sup>13</sup> propose an approach in which security and reliability issues in a distributed computing system are completely separated from one another. However, they suggest that the concepts of "reliability and security are not necessarily best treated so separately, and that their joint consideration can lead to some interesting new insights". In this paper, we propose a protocol where the security of the data is integrated with the data replication process itself.

Herlihy and Tygar present a secure quorum consensus protocol<sup>8</sup> in which they use the concept of a secret *key* for encoding and decoding copies of replicated data. The secret sharing algorithm<sup>16</sup> is used to divide the key into  $n$  pieces, which are distributed on  $n$  sites. To read the object,  $m$  pieces of the key are retrieved to determine its value, and then a read quorum of copies are read and decrypted using the key. To write an object, the new value is encrypted using the key and then distributed to a write quorum of copies. Although this protocol results in secure data, it does so by separating the issue of the security of the key from the

\* This research is supported by the National Science Foundation under grant numbers CCR-8809387 and IRI-8809284.

replication of the data (the encoding of the data is separate from the secret distribution of the key). Our approach deviates from these two approaches<sup>13,8</sup> in the sense that we integrate the security and reliability issues with the protocol itself.

In the next section, we provide some background on security and describe the information dispersal algorithm.<sup>12</sup> In Section 3, we present a data management protocol that integrates the information dispersal algorithm (for *security*) and the quorum protocol (for *reliability*). Although this protocol provides the desired level of security, it does not achieve the same level of availability for both read and write operations as the quorum protocol. In Section 4, we describe a propagation mechanism,<sup>17</sup> which ensures that components of a distributed application share information through implicit communication. This mechanism is integrated with our protocol to achieve the same level of availability for both read and write operations as other quorum protocols, while maintaining the desired level of security. Section 5 concludes the paper.

## 2. SECURITY

In this section, we define the two primary concerns for secure operations on data. Next, we briefly discuss some of the algorithms that provide secure operations in a network. We then describe in greater detail the encoding and decoding techniques used by one of these algorithms.<sup>12</sup>

### 2.1 Background

There are two aspects of security: *confidentiality* and *resiliency*. *Confidentiality* of a file determines the maximum number of representatives of the file that may be accessed without compromising the information stored in the file. *Resiliency* (in the context of security) of a file determines the maximum number of representatives that may be destroyed by an adversary without losing the file completely.

One of the approaches for maintaining a secure file in a distributed system is Shamir's secret sharing algorithm.<sup>16</sup> In this algorithm, a file,  $f$ , is broken into  $n$  representatives,  $f_1, \dots, f_n$ , each of them is the same size as  $f$ , i.e.,  $|f_i| = |f|$ ,  $1 \leq i \leq n$ , where  $|f|$  represents the number of characters in  $f$ . The file  $f$  can be constructed from any  $m$  representatives, but  $m-1$  representatives do not give any information about  $f$ . Thus, if we store the  $n$  representatives of  $f$  at different sites in the network, the algorithm has both desirable aspects of secure data: it provides confidentiality of information in the file (at least  $m$  sites must be broken into in order to access the file), and it is resilient to accidental or intentional information losses (the file can be reconstructed despite a loss of  $n-m$  representatives). However, the algorithm leads to  $n$ -fold increase in total storage.

Rabin<sup>12</sup> proposed the *information dispersal algorithm* (IDA), which attains data security at nominal storage cost. In this algorithm the file can be constructed from any  $m$  representatives but it can not be reconstructed in its entirety from any set of  $m-1$  representatives. An important characteristic of IDA is that each representative is of size  $|f|/m$ . Therefore, the total storage

cost involved is  $(n/m)|f|$ . Since  $n$  and  $m$  can be chosen such that  $n \geq m$  and  $(n/m) \approx 1$ , this dispersal scheme is highly efficient in terms of storage requirements. Furthermore, the algorithm does have both aspects of security: it guarantees confidentiality and can recover from loss of information. Also, the standard replication is a special case of this technique when  $m = 1$ . Rabin discusses several techniques to make the dispersal and reconstruction computationally efficient. We now discuss the details of this algorithm.

### 2.2 Efficient dispersal of information

Consider a file  $f$ , which is a stream of characters and is of size  $L$ , i.e.,

$$f = b_1, b_2, \dots, b_L.$$

Each character  $b_i$  may be considered as an integer taken from a certain range  $[0 \dots B]$ . For example, if  $b_i$  is an eight bit character then  $0 \leq b_i \leq 255$ . We choose a prime number  $p$  such that  $p > B$ . In the case of eight bit characters,  $p = 257$  will suffice; a larger prime could also be chosen. Now,  $f$  can be considered as a string of *residues mod p*, i.e., a string of elements in the finite field  $\mathbb{Z}_p$ . All the following computations are performed in  $\mathbb{Z}_p$ , i.e., all operations are performed in terms of mod  $p$ .

Our goal is to disperse  $f$  such that:

1. With overwhelming probability no more than  $k$  representatives will be lost as a result of failures or security mishaps.
2. With overwhelming probability fewer than  $m$  representatives can be accessed by intruders.

We call  $k$  the *resiliency level* of  $f$ , and  $m$  its *confidentiality level*. We choose  $n$  such that  $n = m + k$ , where  $m$  and  $k$  are determined as described above. We choose  $n$  vectors

$$a_i = (a_{i1}, \dots, a_{im}) \in \mathbb{Z}_p^m, 1 \leq i \leq n$$

such that every subset of  $m$  different vectors are linearly independent. One way of ensuring that any  $m$  vectors out of  $a_1, \dots, a_n$  are linearly independent, is to choose  $n$  different elements  $\alpha_1, \dots, \alpha_n \in \mathbb{Z}_p$  (this requires that  $n < p$ ), and set

$$a_i = (1, \alpha_i, \dots, \alpha_i^{m-1}), 1 \leq i \leq n.$$

### SPLITTING

The file  $f$  is segmented into sequences of length  $m$ ; if the size of  $f$  is not a multiple of  $m$ , it can be padded with null characters to make its size a multiple of  $m$ . Thus, if we assume that the size of  $f$ ,  $L$ , is a multiple of  $m$  then:

$$f = (b_1, \dots, b_m), \dots, (b_{(i-1)m+1}, \dots, b_{im}), \dots, (b_{(L/m-1)m+1}, \dots, b_{(L/m)})$$

We denote the  $i$ th segment of  $f$  as  $s_i$ , where each  $s_i$  is of length  $m$ , i.e.,

$$f = s_1, s_2, \dots, s_{(L/m)}$$

We construct each representative  $f_i$  by picking the  $i$ th vector  $a_i$ , and disperse  $f$  as follows:

$$f_i = a_i \cdot s_1, a_i \cdot s_2, \dots, a_i \cdot s_{(L/m)}$$

Equivalently,  $f_i$  can be written as a string of characters

$$f_i = c_{i1}, c_{i2}, \dots, c_{i(L/m)}$$

where  $c_{ik} = a_i \cdot s_k$ . Hence  $|f_i| = (|f|/m)$ , or  $|f_i| = (L/m)$ .

### RECOMBINATION

If  $m$  representatives of  $f$ , say  $f_1, f_2, \dots, f_m$  are available,  $f$  can be reconstructed as follows. We let  $A = (a_{ij})_{1 \leq i, j \leq m}$  be the  $m \times m$  matrix such that  $i$ th row of the matrix is  $a_i$ . It can be easily verified that the result of multiplying the matrix  $A$  with the segment  $s_i$  of  $f$  are the  $i$ th characters of each  $f_i$ . Hence

$$A \cdot s_i = \begin{bmatrix} c_{1i} \\ \cdot \\ \cdot \\ \cdot \\ c_{mi} \end{bmatrix}$$

Thus, by using the inverse of the matrix  $A$  and the  $i$ th character taken from each representative  $f_i$ , we can extract the  $i$ th segment  $s_i$ . That is:

$$s_i = A^{-1} \cdot \begin{bmatrix} c_{1i} \\ \cdot \\ \cdot \\ \cdot \\ c_{mi} \end{bmatrix}$$

Therefore, from all the characters of  $f_1, \dots, f_m$  we can extract the segments  $s_1, \dots, s_{(L/m)}$ , which are all the segments of the original file  $f$ .

The information dispersal algorithm, considers only the case when a write operation updates  $n$  representatives and a read operation requires  $m$  representatives. This protocol is fault-tolerant for read operations but is not fault-tolerant for write operations. That is, all  $n$  representatives must be available for a write operation to complete successfully. In the next section we combine a quorum based data management protocol with the dispersal algorithm, and propose a generalized algorithm to attain security and reliability of data in a distributed environment.

### 3. A SECURE QUORUM PROTOCOL

We consider a set of *sites* connected by bidirectional links. A *distributed database* consists of a set of *objects*, which may reside at different sites.\* Users execute *transactions* that read and write the objects in the database. The execution of a transaction is *atomic*, i.e., before a transaction terminates it either *commits* or *aborts* all changes it made to the database. We also assume that transaction execution is synchronized by an underlying concurrency control mechanism, e.g., two-phase locking protocol<sup>4</sup> or timestamp ordering protocol.<sup>14</sup>

\* Henceforth, files and other abstract entities will be referred to as objects.

#### 3.1 The protocol

In a distributed system we are interested in achieving a high level of security and reliability of data. The aspect of *resiliency* in the context of security is guaranteed by replication, since an object can be recovered if some of the copies of the object are destroyed or lost. However, replication does not guarantee *confidentiality*. We use Rabin's efficient dispersal technique<sup>12</sup> to distribute the object at  $n$  different sites. In this section we integrate the quorum protocol<sup>6</sup> with the dispersion technique to achieve both security and reliability of data. The information dispersal algorithm (IDA) only considers the fault-tolerance aspects of read operations on objects. Our approach extends IDA, and makes both read and write operations fault-tolerant. An interesting outcome is that the protocol described here can achieve the same level of availability as the quorum protocol for at least one operation (read or write), while providing the desired level of security.

Given an object  $x$ , we distribute it on several sites using the dispersal technique such that it guarantees the *confidentiality level* of  $m[x]$  and the *resiliency level* of  $k[x]$ . The total number of representatives is  $n[x]$ , where  $n[x] = m[x] + k[x]$ . Each representative is stored at a different site in the network. A representative consists of the encoded data and the associated encoding vector. The properties guaranteed by the dispersal technique are:

**Storage property.** The total storage involved in storing the object  $x$  is  $(n[x]/m[x])|x|$ , where  $|x|$  is the size of the object.

**Confidentiality property.** The object  $x$  can be constructed from any  $m[x]$  representatives of  $x$  stored at  $n[x]$  sites, but  $m[x] - 1$  representatives of  $x$  are not sufficient to reconstruct the entire object  $x$ .

We associate with each representative a *version number*, which is initialized to one, and with each object  $x$  a *read quorum*,  $q_r[x]$ , and a *write quorum*,  $q_w[x]$ . A read operation,  $r[x]$ , is executed as follows:

- (1) Select  $q_r[x]$  representatives of  $x$ , and determine the maximum version number,  $vn_{\max}$ , of the selected representatives.
- (2) Read  $m[x]$  representatives with version number  $vn_{\max}$  in the read quorum, and reconstruct  $x$  by employing the recombination technique.

A write operation,  $w[x]$ , is executed as follows:

- (1) Select  $q_w[x]$  representatives of  $x$ , and determine the maximum version number,  $vn_{\max}$ , of the selected representatives.
- (2) Create  $q_w[x]$  representatives out of  $n[x]$  using the splitting technique.
- (3) Write these  $q_w[x]$  representatives with version number  $vn_{\max} + 1$ .

Read and write quorums must satisfy the following requirements:

$$m[x] \leq q_r[x] \leq n[x] \quad (4.1)$$

$$\max \left( m[x], \left\lceil \frac{n[x] + 1}{2} \right\rceil \right) \leq q_w[x] \leq n[x] \quad (4.2)$$

$$n[x] + m[x] \leq q_r[x] + q_w[x] \leq 2 \cdot n[x] \quad (4.3)$$

Equation (4.1) captures the requirement that each read operation must access at least  $m[x]$  representatives, otherwise the confidentiality property indicates that the object cannot be reconstructed completely.

Equation (4.2) places two restrictions on the lower bound of the write quorums. First, a write operation on object  $x$  must at least write  $m[x]$  representatives. If a write operation writes fewer than  $m[x]$  representatives, subsequent read operations will not be able to construct  $x$ . Second, any two write operations of an object  $x$  must have a non-empty intersection, i.e., there must be at least one representative written by both operations. This restriction is imposed because every write operation must assign a new version number greater than the version numbers assigned to any representative. Note that the second restriction on the lower bound is the same as the one imposed on write operations in the quorum protocol.

Finally, equation (4.3) imposes the restriction that for an object  $x$ , any two sets of sizes  $q_r[x]$  and  $q_w[x]$  must contain at least  $m[x]$  representatives in common. Since a read operation intersects with every write operation, it can determine the highest version number written. Furthermore, the entire object  $x$  can be constructed from  $m[x]$  representatives with the highest version number. The next theorem formalizes these arguments and shows the necessity of the lower bound in equation (4.3).

### Theorem 1

For a read operation,  $r[x]$ , to read the current value of an object  $x$   $q_r[x] + q_w[x]$  must be greater than or equal to  $n[x] + m[x]$ .

*Proof.* If  $q_r[x] + q_w[x] < n[x] + 1$ , then in the worst case  $r[x]$  may not access any representative with the highest version number. Hence, we only have to consider the case when:

$$n[x] < q_r[x] + q_w[x] < n[x] + m[x]$$

i.e., the case where a read and a write operation have a non-empty intersection that does not contain  $m[x]$  representatives. In this case,  $r[x]$  intersects with every write operation at least at one representative, and hence,  $r[x]$  can determine the value of the highest version number  $vn_{\max}$ . Let  $w_{\max}[x]$  be the write operation that writes  $x$  with the version number  $vn_{\max}$ . Let  $q_r[x] + q_w[x] = n[x] + m[x] - 1$ . Then, in the worst case, operations  $w_{\max}[x]$  and  $r[x]$  have at most  $m[x] - 1$  representatives in common. But by the confidentiality property, this implies that  $r[x]$  cannot reconstruct the entire object  $x$  with  $vn_{\max}$ . Thus,  $q_r[x] + q_w[x]$  must be greater than or equal to  $n[x] + m[x]$ .  $\square$

We next compare equations (4.1), (4.2) and (4.3) with the equations from the quorum protocol relating the read and write quorums of replicated objects. In the quorum protocol, a read operation can be performed by accessing as few as one copy of an object. On the other hand, in our protocol, one representative does not represent the entire object, and hence from Equation (4.1) a read operation must access at least  $m[x]$  representatives. It must be noted, however, that

\* Two operations conflict if they operate on the same object and at least one of them is a write operation.

this is desirable to ensure the *confidentiality* of the information contained in the object  $x$ .

A similar distinction exists for the write operations in the two protocols. In the quorum protocol, a write operation can be executed with as few as  $\lceil (n[x] + 1)/2 \rceil$  copies of an object  $x$ , while our protocol requires at least the maximum of  $m[x]$  and  $\lceil (n[x] + 1)/2 \rceil$  representatives. This restriction, however, can be made void by always choosing a value of  $m[x]$  such that  $m[x] \leq (n[x]/2)$ . Hence, equation (4.2) requires that two write operations must have at least one representative in common, which is the same as in the quorum protocol. This is due to the observation that in order to execute a write operation the current value of the object is not necessary, rather the highest version number associated with any of its representatives must be available.

Finally, in the quorum protocol, a read and a write operation on an object  $x$  need only have one copy in common while equation (4.3) requires  $m[x]$  representatives to be in common. Note that for the purpose of correctness, i.e., to ensure *one-copy serializability*,<sup>2</sup> the sum of read and write quorums of  $n[x] + 1$  representatives would be sufficient. This is due to the fact that to ensure one-copy serializability, all our protocol has to guarantee is that two conflicting operations\* must physically conflict on at least one representative. Since our protocol imposes stronger restrictions on read and write quorums, it must ensure one-copy serializability. We next analyze the level of availability achieved by our protocol.

### 3.2 Resiliency of the protocol

This simple protocol is a generalization of the quorum protocol in which objects are not split and dispersed, or equivalently in which for object  $x$ ,  $m[x] = 1$ . In this section we compare the levels of resiliency achieved by both protocols: we show that in general our protocol can achieve the same resiliency level for at least one operation while providing a higher level of security. We start by formalizing the notion of resiliency as follows: an implementation of an object  $x$  has *read resiliency*,  $R_r[x]$ , if a read operation on  $x$  can be executed even after  $R_r[x]$  representatives are inaccessible due to site or partitioning failures. *Write resiliency*,  $R_w[x]$ , for an object  $x$  is defined similarly.

For purposes of comparison, let  $Q_w[x]$  and  $Q_r[x]$  be the read and write quorums associated with an object  $x$  according to the quorum protocol, and let  $N[x]$  be the total number of copies implementing object  $x$ . This implementation has a read resiliency  $R_r[x] = N[x] - Q_r[x]$  and a write resiliency  $R_w[x] = N[x] - Q_w[x]$ . We now present two implementations using our protocol, one that achieves the same write resiliency as that achieved by the quorum protocol (but a lower degree of read resiliency), and another that achieves the same read resiliency (but a lower write resiliency). Both implementations provide the same level of security attained by IDA while making read or write operations fault-tolerant.

We implement our protocol using  $n[x] = N[x]$  representatives, and any value of  $m[x]$  such that  $m[x] \leq (n[x]/2)$ . The dispersion approach can achieve the same write resiliency for write operations by assigning

$q_w[x] = Q_w[x]$ ; since  $n[x] = N[x]$ ,  $q_w[x] > (N[x]/2) = (n[x]/2)$ , and since  $m[x] \leq (n[x]/2)$ ,  $q_w[x] > m[x]$ . Unfortunately, to achieve this degree of write resiliency and communication cost, read operations in our protocol become more expensive and less resilient to failures. More specifically,  $q_r[x] = Q_r[x] + m[x] - 1$ , i.e., the read quorum has increased in size by  $(m[x] - 1)$ , and hence the read resiliency has decreased by that factor too. Our second implementation achieves the same read resiliency as the quorum protocol but at the expense of write operations. Let  $n[x] = N[x]$  and  $q_r[x] = Q_r[x]$ , thus achieving the same read resiliency as the quorum protocol. However, write quorums are  $q_w[x] = Q_w[x] + (m[x] - 1)$  in this case, i.e., the write quorum has increased in size by  $(m[x] - 1)$ , and thus lowering the write resiliency of  $x$ . Note that from the storage property the storage cost is  $n[x]/m[x]$  and  $m[x]$  is generally greater than one in case of secure data. Therefore, a side-effect of using IDA is that our protocol requires less storage for data replication.

In conclusion, we note that since a read and a write quorum for an object  $x$  must contain  $m[x]$  representatives, both read and write operations cannot achieve the same degree of resiliency and communication cost as the quorum protocol while maintaining the desired level of security. In the next section we present a special mechanism that overcomes this problem.

#### 4. A HIGHLY AVAILABLE SECURE QUORUM PROTOCOL

In the previous section we showed that in order to achieve *confidentiality* of an object  $x$  to  $m[x]$  and to achieve *resiliency* of  $x$  to  $n[x] - m[x]$ , the size of the intersection between read and write operations increases from one copy of  $x$  to  $m[x]$  representatives of  $x$ . The larger size intersection results in increased communication costs for read and/or write operations. In this section we provide an underlying mechanism to ensure that the information written by a write operation on an object is eventually propagated to all representatives of the object in the system. Although all representatives of an object  $x$  are updated as a result of a write operation on  $x$ , it does not mean that  $q_w[x]$  in this protocol is  $n[x]$ . By using this underlying mechanism, we will show how to decrease the size of the intersection from  $m[x]$  representatives to one representative while maintaining the desired security of  $x$ . First, we describe the underlying mechanism to propagate write operations to all representatives of an object. Next, we explain how to integrate our protocol with the propagation mechanism. Finally, we compare our modified protocol with the quorum protocol and demonstrate that we achieve the same level of resiliency and communication cost for read and write operations in our protocol.

##### 4.1 The propagation mechanism

A common technique to propagate information efficiently in a network and, thus, synchronize various components of a distributed application is to construct a *log* of certain application specific events that have occurred in the network.<sup>15</sup> In the case of a replicated database such events include reading or writing a copy of an object at the coordinator site of a transaction.

Each site maintains a local copy of the log, which is organized as an ordered sequence of event records, and a propagation mechanism is employed to keep the copies of the log up-to-date. The mechanism makes use of communication operations, *send* and *receive*, to exchange portions of the copies of the log for this purpose.<sup>5, 7, 9, 11, 7</sup> The background messages used in the propagation mechanism to bring all the copies of the log up-to-date are also referred to as gossip messages.<sup>11</sup> We have chosen the algorithm proposed by Wu and Bernstein<sup>17</sup> to integrate the propagation mechanism with our protocol.

Wu and Bernstein<sup>17</sup> describe an efficient implementation of the propagation mechanism. Each site,  $S_i$ , maintains a time-table,  $T_i$ , which is an  $N \times N$  array of timestamps of events that have occurred in the network, where  $N$  is the total number of sites. A site uses the time-table to place a bound on how out-of-date other sites are about events that have happened in the network. The time-table allows a site to decide what portion of its copy of the log should be sent to another site, and when all sites have learned about a particular event. The latter information is used by the site to determine when certain portions of its copy of the log can be discarded. Hence, a site retains a particular event record in its copy of the log only if it is not certain that all other sites have learned of that event. The *happened-before* relation, " $\rightarrow$ ",<sup>10</sup> relates the application specific events and the communication operations employed by the propagation mechanism. Periodically a site sends its timetable and a portion of its copy of the log to another site. On receiving such a message, a site updates its copy of the log by including event records of which it was unaware and updates its time-table using information in the received time-table. The following two properties are guaranteed by the algorithm:

**Propagation property.** Every site eventually learns of each event.

**Causality property.** If  $e_1$  and  $e_2$  are two events such that  $e_1 \rightarrow e_2$ , then if a site knows of  $e_2$ , it must also know of  $e_1$ .

The propagation property is dependent on the assumption that site failures and network partitions are not permanent. It follows from the causality property that a site can process events in the *happened-before* order.

It should be noted that in the model of the system discussed above, all communications among sites is performed implicitly by exchanging the copies of logs among the sites. That is, explicit communication operations, *send* and *receive*, are not available to application programs. Instead, an application program relies on the underlying propagation mechanism to inform other sites about its operation request (for example, a find operation on a distributed dictionary). The responses of other sites (for example, the results of a find operation on a distributed dictionary) are also communicated to the application program through the log. Although the propagation mechanism has an overhead of maintaining copies of the log, it has several advantages that offset this extra overhead: it can be easily implemented in an unreliable network, and the number of messages in the system can be reduced at the expense of the size of messages. Furthermore, the size of the copies of the log is bounded since sites discard event records from their

copies as soon as they discover that all sites have learned about the events corresponding to these event records. Several optimizations have been proposed to reduce the overhead associated with this mechanism.<sup>17</sup>

#### 4.2 The integrated protocol

We now integrate the propagation mechanism introduced in the previous subsection with the execution of read and write operations of transactions. The dispersion technique described earlier is used to store the representatives of objects at different sites. The model of the system remains the same as developed in Section 3.1 except for the distinction that application programs, transactions in our case, do not explicitly communicate with *remote* sites in the system. All communication is achieved by modelling operations and the results of the operations as events in the system, and then exchanging the copies of the log among the sites. The site, where a transaction originates, is designated as the *coordinator* of the transaction. Read and write operations of a transaction are recorded as *events* in the copy of the log at the coordinator; other sites in the network learn about these events as a result of the propagation. Furthermore, sites agreeing to be in the quorum of an operation do not communicate explicitly with the coordinator. Instead, their decision to be in the quorum is also recorded as an *event* in their copy of the log; they too rely on the underlying communication operations to propagate these events to the coordinator.

We associate with each object  $x$ , a *read quorum*,  $q_r[x]$ , and a *write quorum*,  $q_w[x]$ . A read operation,  $r[x]$ , is executed as follows:

- (1) A read operation,  $r[x]$ , results in an event,  $r[x]$ -event, at the coordinator. An  $r[x]$ -event record is placed in the coordinator's copy of the log. When the coordinator's copy of the log is propagated to other sites, the effect is the same as the transaction sending a read request to other sites in the system.
- (2) When a site,  $S$ , learns of  $r[x]$ -event and decides\* to be in the quorum for  $r[x]$ , an event,  $ok_S(r[x])$ -event, occurs at that site. The event record corresponding to  $ok_S(r[x])$ -event in the copy of the log includes the value of the representative of  $x$  at that site. When the site's copy of the log is eventually propagated to the coordinator, the effect is the same as the transaction receiving a reply to the read request from a site in the quorum.
- (3) The operation,  $r[x]$ , is not completed until the coordinator can determine that  $q_r[x]$  representatives of  $x$  have been accessed. The events,  $ok_S(r[x])$ -event, are observed at the coordinator for this purpose. After accessing  $q_r[x]$  representatives of  $x$ , the coordinator returns the value of  $x$  to the requesting transaction. The object  $x$  is constructed by identifying  $m[x]$  representatives with the highest version number, and by using the recombination technique described in Section 2.2.

A write operation,  $w[x]$ , is executed as follows:

- (1) A write operation,  $w[x]$ , results in an event,  $v[x]$ -event, at the coordinator. A  $v[x]$ -event record is

\* This decision is based on the concurrency control mechanism employed.

placed in the coordinator's copy of the log. When the coordinator's copy of the log is propagated to other sites, the effect is the same as the transaction sending a version request to other sites in the system.

- (2) When a site,  $S$ , learns of  $v[x]$ -event and decides to be in the quorum for  $w[x]$ , an event,  $ok_S(v[x])$ -event, occurs at that site. The event record corresponding to  $ok_S(v[x])$ -event in the copy of the log includes the version number of the representative of  $x$  at that site. When the site's copy of the log is eventually propagated to the coordinator, the effect is the same as the transaction receiving a reply to the version request from a site in the quorum.
- (3) The operation,  $w[x]$ , is not completed until the coordinator can determine that  $q_w[x]$  representatives of  $x$  have been accessed. The events,  $ok_S(v[x])$ -event, are observed by the coordinator for this purpose. The operation,  $w[x]$ , is completed when an event,  $w[x]$ -event, occurs at the coordinator. A  $w[x]$ -event record includes the new value and the new version number of  $x$  that will be used to update the representatives of  $x$  at various sites in the network. When the coordinator's copy of the log is propagated to other sites, the effect is the same as the transaction sending a write request to other sites in the system. Note that the underlying concurrency control mechanism ensures that no other transaction can access  $x$  until the transaction executing  $w[x]$  commits or aborts at its quorum.

The modified protocol must satisfy the following requirements:

$$m[x] \leq q_r[x] \leq n[x] \quad (5.1)$$

$$\max \left( m[x], \left\lceil \frac{n[x] + 1}{2} \right\rceil \right) \leq q_w[x] \leq n[x] \quad (5.2)$$

$$n[x] + 1 \leq q_r[x] + q_w[x] \leq 2 \cdot n[x] \quad (5.3)$$

Equations (5.1) and (5.2) are the same as Equations (4.1) and (4.2) from the previous section; this is due to the identical considerations. On the other hand, Equation (5.3) is different and states that the read and write quorum intersection of one representative is sufficient. This is a significant improvement from the previous section, where it was required that a read and a write quorum for an object  $x$  must have an intersection of  $m[x]$  representatives. This is due to the fact that the event record for  $w[x]$  in the copy of the log at a site contains the entire information about  $x$ , and not only the information concerning the representative of  $x$  at that site. Since  $q_r[x] + q_w[x] \geq n[x] + 1$ , there will always be at least one representative with the highest version number in  $q_r[x]$  corresponding to some write operation  $w_{\max}[x]$ . If  $r[x]$  collects  $m[x]$  representatives or more with the highest version number, it can reconstruct the entire object  $x$ . In the case when  $r[x]$  collects fewer than  $m[x]$  representatives with the highest version number, then  $w_{\max}[x]$ -event has not been propagated to all representatives of  $x$  in the network. Since an event record is discarded by a site only when all sites learn about the event,  $w_{\max}[x]$ -event record must still exist in the log. Hence, the coordinator can always construct the entire object  $x$  by using its copy of the log. This is formally proved in the following theorem.

**Theorem 2**

For a read operation,  $r[x]$ , to read the current value of an object  $x$ ,  $q_r[x] + q_w[x]$  must be greater than or equal to  $n[x] + 1$ .

*Proof.* Since  $q_r[x] + q_w[x] \geq n[x] + 1$ ,  $r[x]$  intersects with every write operation at least at one representative, and hence, can determine the highest version number  $vn_{\max}$ . Let  $w_{\max}[x]$  be the write operation that writes  $x$  with version number  $vn_{\max}$ . Since  $q_r[x] \geq m[x]$ , there are two cases to consider. If  $r[x]$  collects  $m[x]$  representatives or more with  $vn_{\max}$  in its quorum, it can construct  $x$  from the replies of the sites in the quorum. Otherwise,  $r[x]$  collects fewer than  $m[x]$  representatives with  $vn_{\max}$  and  $w_{\max}[x]$ -event must exist in the copies of the log at the sites that have the representatives of  $x$  with  $vn_{\max}$  (this is from Ref. 17 where an event record is discarded by a site if it can determine that all sites have learned about the event). At least one of these sites,  $S$ , with  $vn_{\max}$  must be in  $q_r[x]$ . Since at  $S$ ,  $r[x]$  will read from  $w_{\max}[x]$ , then:

$$w_{\max}[x]\text{-event} \rightarrow r[x]\text{-event} \rightarrow ok_S(r[x])\text{-event}$$

Hence by the causality property, when the coordinator of  $r[x]$  learns about  $ok_S(r[x])\text{-event}$ , it must also become aware of  $w_{\max}[x]\text{-event}$ . Thus, the coordinator will always return the current value of  $x$ .  $\square$

**4.3 Resiliency of the protocol**

The quorum intersection requirement in our protocol is identical to that in the quorum protocol, and therefore, our protocol achieves the same level of availability while providing security. As in Section 3.2, let  $Q_w[x]$  and  $Q_r[x]$  be the read and write quorums associated with an object  $x$  according to the quorum protocol, and let  $N[x]$  be the total number of copies implementing object  $x$ . This implementation has a read resiliency  $R_r[x] = N[x] - Q_r[x]$  and a write resiliency  $R_w[x] = N[x] - Q_w[x]$ . In our implementation, we use  $n[x] = N[x]$ ,  $q_r[x] = Q_r[x]$ , and  $q_w[x] = Q_w[x]$ . We choose  $m[x]$  such that:

$$m[x] \leq q_r[x] \leq n[x] \quad (5.1)$$

$$\max \left( m[x], \left\lceil \frac{n[x] + 1}{2} \right\rceil \right) \leq q_w[x] \leq n[x] \quad (5.2)$$

If we choose a value of  $m[x]$  such that  $m[x] \leq (n[x]/2)$  then equation (5.2) reduces to:

$$\left\lceil \frac{n[x] + 1}{2} \right\rceil \leq q_w[x] \leq n[x] \quad (5.2a)$$

which imposes the same restriction on write operations as in the quorum protocol. Thus the only restriction imposed by our scheme is that the read quorum must be greater than or equal to  $m[x]$ . This implies that the range of read quorum assignment is restricted at the lower end in our protocol. However, this is not a serious shortcoming, since any fault-tolerant implementation of an object  $x$  will rarely use the lower end of the read quorum assignments (for write operations to tolerate the failures of  $t$  copies, the read quorum must be greater than  $t$ ). Furthermore, in order to guarantee confidentiality of  $x$ , we will generally choose a value of  $q_r[x]$

that is greater than one, and use this to choose the value of  $m[x]$ . Thus, the restricted range of read quorum assignments does not have any significant ramifications. Note that the storage requirement for replication in  $n[x]/m[x]$  of the size of  $x$  in our modified protocol.

**5. CONCLUSION**

In this paper, we presented a secure and fault-tolerant protocol. We first integrated the information dispersal algorithm<sup>12</sup> with the quorum protocol<sup>6</sup> to provide both security and fault-tolerance in a database system. This resulted in a protocol that extends the fault-tolerant aspects of IDA to both read and write operations. However, this protocol can not achieve the same level of availability of read and write operations as the quorum protocol, while maintaining the desired level of security. In order to overcome this drawback, we integrated the propagation technique<sup>17</sup> with our protocol. The propagation mechanism does have an extra overhead of maintaining copies of the log in the system. However, there are several advantages of this mechanism that offset this overhead: it can be easily implemented in an unreliable network, and the number of messages can be reduced at the cost of the size of messages. We demonstrated that for any read quorum greater than one, our protocol can provide security and reduce the storage while maintaining the same availability for read and write operations. The large size of read quorum is desirable for highly secure databases. Furthermore, in most fault-tolerant systems based on the quorum approach, the write quorum is generally less than all copies of an object, and hence the read quorum is greater than one. Such fault tolerant systems can benefit from the approach proposed in this paper to provide secure database operation and also reduce storage requirements.<sup>1</sup>

**REFERENCES**

1. D. Agrawal and A. El Abbadi, Reducing Storage for Quorum Consensus Algorithms, *Proceedings of the Fourteenth International Conference on Very Large Data Bases* pp. 419-430 (1988).
2. P. A. Bernstein and N. Goodman, The failure and recovery problem for replicated databases, *Proceedings of the Second ACM Symposium of Principles of Distributed Computing* pp. 114-122 (1983).
3. S. Davidson, H. Garcia-Molina and D. Skeen, Consistency in Partitioned Networks, *Computing Surveys* 17 (3) pp. 341-370 (1985).
4. K. P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, The notion of consistency and predicate locks in database system. *Communications of the ACM*, 19 (11) pp. 624-633 (1976).
5. M. J. Fischer and A. Michael, Sacrificing serializability to attain high availability of data in an unreliable network, *ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* pp. 70-75 (1982).
6. D. K. Gifford, Weighted voting for replicated data, *Proceedings of the seventh ACM symposium on operating systems principles* pp. 150-159 (1979).
7. A. Heddaya, M. Hsu and W. Weihl, Two phase gossip: Managing distributed event histories, To appear in *Information Sciences: An international journal*.
8. M. P. Herlihy and J. D. Tygar, How to make replicated data secure, *Advances in Cryptology, Proceedings of*

- CRYPTO '87*, Springer Verlag, *Lecture Notes in Computer Science*, No. 293 pp. 379–391 (1987).
9. T. A. Joseph and K. P. Birman, Low cost management of replicated data in fault-tolerant distributed systems, *ACM Transactions on Computer Systems* **4** (1) pp. 54–70 (1986).
  10. L. Lamport, Time, clocks and ordering of events in a distributed system, *Communications of the ACM* **21** (7) pp. 558–565 (1978).
  11. B. R. Liskow and R. Ladin, Highly available services in distributed systems, *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing* pp. 29–39 (1986).
  12. M. O. Rabin, Efficient dispersal of information for security, load balancing, and fault tolerance, *Journal of the ACM*, **36** (2) pp. 335–348 (1989).
  13. B. Randell and J. Dobson, Reliability and security issues in distributed computing systems, *Proceedings of the Fifth Symposium on Reliability in Distributed Software and Database Systems*, pp. 113–118 (1986).
  14. D. P. Reed, Implementing atomic actions on decentralized data, *ACM Transactions on Computer Systems*, **1** (1) pp. 3–23 (1983).
  15. F. B. Schneider, Synchronization in distributed programs, *ACM Transactions on Programming Languages and Systems* **4** (2) pp. 125–148 (1982).
  16. A. Shamir, How to share a secret? *Communications of the ACM*, **22** (11) pp. 612–613 (1979).
  17. G. T. J. Wu and A. J. Bernstein, Efficient solutions to the replicated log and dictionary problems, *Proceedings of the Third ACM Symposium on Principles of Distributed Computing* pp. 233–242 (1984).

## Announcements

2–6 NOVEMBER 1992

MADRID, SPAIN

### 12th World Computer Congress IFIP Congress '92

The International Federation for Information Processing (IFIP) and the Federación Española de Sociedades de Informática (FESI) invite you to participate in IFIP Congress '92, the 12th World Computer Congress.

#### IFIP

The International Federation for Information Processing is a multinational federation of professional and technical organizations (or national groupings of such organizations) concerned with information and computer sciences. As of March 1989, there are 46 organizational members of IFIP, representing 64 countries.

The aims of IFIP are to promote information science and technology by:

- fostering international cooperation in the field of information processing;
- stimulating research, development and the application of information processing in science and human activity;
- furthering the dissemination and exchange of information about the subject;
- encouraging education in information processing.

IFIP came into existence in January, 1960. It was established after the first International Conference on Information Processing which was held in Paris in June, 1969, under the sponsorship of UNESCO. Its technical work is managed by nine Technical Committees, divided into a number of Working Groups.

#### The World Computer Congresses

The most significant event in the IFIP programme is the World Computer Congress, held every three years. The Congress is an international occasion which attracts information and computer professionals from all over the world, to learn and exchange ideas with their colleagues from other countries.

#### Congress Site

IFIP Congress '92 will be held at the City Hall Pavillions in the Casa de Campo, the largest park in Madrid, just a few minutes ride by underground or bus from the centre of the old town. Parallel sessions will take place at nearby buildings, amidst the most picturesque landscape of Spain's capital city.

#### Technical Programme

IFIP Congresses' technical programmes provide, in plenary and multiple parallel sessions of outstanding quality, an up-to-date state-of-the-art panorama of information and computer science and technology disciplines. Managers, professionals, scientists and educators can choose in them the particular menus that cover their specific areas of interest.

#### Exhibition

In 1992, Spain's major annual international exhibition for the information and computer industries (SIMO) will be held in Madrid in conjunction with the World Computer Congress.

SIMO was started in 1961 and has grown since then to cover a surface of 30,800 sq m, with over 500 exhibitors and almost 180,000

visitors, mostly professionals coming from all the country and abroad.

#### Travel, Accommodation and Social Programme

The official air carrier for the Congress is Iberia, the Spanish National Air Company. The international airport of Madrid (Barajas) is one of the six busiest airports in Europe, with frequent connections to all countries. Spain is a member of the European Community and maintains diplomatic relations with all IFIP member nations and, in fact, with almost all others. Information about passport and, in some cases, visa requirements can be obtained at any Spanish diplomatic delegation.

Accommodation has been arranged in several categories, from de luxe through economy, in order to suite the needs of all delegates.

Besides the Congress receptions and official banquet, an extensive sightseeing, cultural and activities programme is being planned for delegates and accompanying persons. Pre- and post-congress tours will be available for those wishing to explore further into the many opportunities that Spain offers to suit any visitor's interest.

#### Additional Information

The official language of the Congress will be English. Simultaneous translation into Spanish will be offered by sponsoring entities.

For other additional information, contact us at:

IFIP Congress '92, FESI, Federación Española de Sociedades de Informática, Hortaleza 104-2.º Izqda, 28004 Madrid, Spain. (electronic mail: fesi @ dit.upm.es)