Coding-Based Replication Schemes for Distributed Systems

Gagan Agrawal and Pankaj Jalote, *Senior Member, IEEE*

***Abstract*— Data is often replicated in distributed systems to improve availability and performance. This replication is expensive in terms of disk storage since the existing schemes generally require full files to be stored at each site. In this paper, we present schemes which significantly reduce the storage requirements in replication based systems. These schemes use the coding method suggested by Rabin to store replicated data. The first scheme that we present is a modification of the simple voting algorithm and its quorum requirements. We then show how some of the extensions of the voting algorithm can also be modified to get storage efficient schemes for managing such replication. We evaluate the availability offered by these schemes and show that the storage space required to achieve certain availability are significantly lower than the conventional schemes with full file replication. Since coding is used, these schemes also provide a high degree of data security.**

***Index Terms*— Availability, coding schemes, data replication, data security, disk usage, distributed databases, fault-tolerance, performance evaluation, voting protocols.**

I. Introduction

IN A DISTRIBUTED SYSTEM, data can be replicated to provide fault-tolerance against site failures and network partitions and to improve performance. This data replication requires a replica control algorithm to maintain consistency of the data. A survey of such methods can be seen in [12].

One such method for replica control is the weighted voting scheme suggested by Gifford [14]. In this algorithm, each node is assigned a number of votes. If N is the total number of votes assigned to all the nodes, then a quorum of r votes is required to do a read operation, and a quorum of w votes is required to perform a write operation. These quorum values are such that, $r + w > N$ and $2 * w > N$. This ensures write-write mutual exclusion and read-write mutual exclusion. A particular case of the weighted voting is *simple voting* in which each node is assigned exactly one vote.

With weighted voting, a node which wants to perform an operation on data, first sends a request to all the nodes in the system. The nodes reply with their version numbers (denoting the number of successful updates made). When the requester node gets $r(w)$ votes, it can perform the read (write) operation. For read, it reads the data from node with the highest of the version numbers given by the nodes that have replied. For a write, the node determines the highest version number, and writes on all the nodes that constitute the write quorum. All of these nodes then have the version number as one more than the highest version number existing in the system previously. The quorum conditions ensure that the node with the highest of the version numbers in a read/write quorum has the latest copy of the data. Many extensions to the voting method have been proposed [4], [11], [13], [19]–[21], [25]. Performance and reliability issues of these schemes have been studied in [5], [7], [8].

A major drawback of replication based schemes is the high degree of disk storage requirement. If the file is replicated at N nodes, the disk storage requirement increases N fold. Just for maintaining availability of data against failure of one node at a time, the data has to be replicated at 3 nodes [21], incurring three times the disk storage costs.

In this paper, we present schemes for maintaining replicated files, that significantly reduce the amount of storage space required to maintain the files with a given availability. These schemes use the coding suggested by Rabin [23], [24] to store the files. In this coding a file F is encoded and broken into n parts, each of size $|F|/m$ (m and n are parameters such that $m \leq n$). Only one such part of the file, called a coded partial file (CPF), is stored at each node. The file can be reconstructed by any m of such n CPF's. With the files being stored in a coded form, the read and write algorithms have to be redesigned. The first scheme that we present, simple voting with coding (SVWC), is obtained by modifying the simple voting algorithm and the quorum requirements. We also present how dynamic voting [17] is modified to obtain dynamic voting with coding (DVWC). We also discuss how some of the other extensions to the voting algorithm, weighted voting [14] and the approach of coteries [13], can also be modified.

Use of this coding in storing replicated data also provides a high degree of data security [3]. Since the file is stored in a coded form and only $1/m$th fraction of the file is stored at one nodes. An unauthorized user has to break into at least m different nodes and decipher the coding in order to read the file. This ensures far higher security of data as compared to the storage of entire file at each node, with or without any coding.

The rest of the paper is organized as follows. We briefly review some other approaches to reducing storage in replication based systems in Section II. We discuss the coding scheme as suggested by Rabin in Section III, we look into how it may be used to store replicated data and our first scheme (SVWC) in

Manuscript received April 1992; revised November 1992.
G. Agrawal is with the Department of Computer Science, University of Maryland, College Park, MD 20742 USA.
P. Jalote is with the Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India.
IEEE Log Number 9408134.

Section IV. The performance (availability and communication overheads) of SVWC is evaluated in Section V. We consider DVWC in Section VI and some other extensions to the voting scheme in Section VII.

II. Related Work

In this section, we briefly review some other approaches to reducing storage requirements in replication based systems. The only other known approaches to reducing storage requirements in replication based schemes are voting with witnesses suggested by Paris [19], [20] and the fragmentation based approach suggested by Agrawal and Abbadi [1], [2].

In Voting with Witnesses, some of the regular copies are replaced by witnesses. Witnesses store only the version number for that site but no data. They, therefore, occupy very little space. Witnesses can participate in read and write operations by sending their version number but read or write quorum must include at least one regular copies with the latest update. In [19], it has been shown that replacing some of the regular copies with witnesses has only marginal effect on availability, while reducing the storage requirements. However, this reduction in storage requirements is marginal since if many regular copies are replaced by witnesses, then the availability does get effected severely.

In the fragmentation based approach, a file to be replicated is broken into N parts, called *fragments*, where N is the number of sites participating in the replication. m fragments $(m < N)$ are stored at each site such that each fragment is stored at exactly m sites. With this, the storage has been reduced to m/N of the original requirement. Any $N - m + 1$ sites will always suffice to reconstruct the original file. This approach, as such, provides very low resiliency since the read and write quorums must intersect at $N - m + 1$ sites. A propagation based mechanism is used to improve resiliency in this scheme. In this, the transactions are stored in logs till other sites in the system can recover. With this, the read and write quorum need to intersect at just one site (same as with simple voting). Even with this, the maximum reduction in the storage space that this scheme can provide is restricted to $N/\lceil N + 1/2 \rceil$. Further, this approach has many deficiencies. With the complicated propagation based mechanism, the conceptual simplicity of voting based methods is lost. Further, the logs used in this mechanism take storage space themselves. This becomes significant if the failures are frequent and the recoveries do not take place fast enough. The propagation based approach incurs significant communication overheads as well.

This scheme is also incapable of accommodating a change in the number of sites participating in the replication which might occur when the system is operational. This is because when the file is fragmented, the number N gets fixed. If the number of sites participating in the replication increases (or decreases) then the number of fragments into which the file has to be fragmented also changes. Also, the way this fragmentation is done, this is likely to incur significant overheads if during a write operation, the size of the file gets changed significantly, since the fragments will need to be created again.

III. Coding Scheme

In the coding scheme suggested by Rabin [23], [24], a file F of size $|F|$ is broken into n parts of size $|F|/m$ such that any m of these n parts are sufficient to reconstruct the file, where, $n \geq m$.

In this scheme, splitting and recombining files is done by using n vectors in m dimensional vector space, such that any m of these n are linearly independent. We will refer to such a system of coding as an (m, n) system. To understand this coding, consider a matrix FM, in which the bytes of the file are packed row-wise, m bytes in each row. In all the computations described, a byte is an integer in the range $0 \cdots 2^l - 1$, (l is the number of bits in a byte). All operations are in modulo p, for a prime integer $p > 2^l$.

Let the n vectors of dimension m chosen for splitting and recombining be

$$a_i = a_{i1} \cdots a_{im}, \qquad i = 1 \cdots n.$$

A splitting matrix SM is constructed by arranging these n vectors column-wise. That is, $\mathrm{SM}_{ij} = a_{ji}$. A coded file matrix CF is obtained as follows

$$\mathrm{FM} * \mathrm{SM} = \mathrm{CF}.$$

The ith column of CF represents the fragment to be identified as the ith Coded Partial File (CPF). The way in which these CPF's have been coded, a group of m consecutive bytes in the original file is represented as one byte in each of the n CPF's. This procedure of splitting files requires in all n additions and n multiplications per byte of the original file. For recombining, suppose that the m CPF's available are $r_1, r_2, \cdots, r_m$. To obtain the original file, we construct a matrix SM$'$, such that $\mathrm{SM}'_{ij} = \mathrm{SM}_{ir_j} = a_{r_j i}$. That is, the column j of SM$'$ is the column r_j of SM. SM$'$ is an invertible $m * m$ matrix because its m columns are linearly independent m dimensional vectors. We further construct a matrix CF$'$, representing the m CPF's available for computing the original file, such that $\mathrm{CF}'_{ij} = \mathrm{CF}_{ir_j}$. Clearly

$$\mathrm{FM} * \mathrm{SM}' = \mathrm{CF}' \Rightarrow \mathrm{FM} = \mathrm{CF}' * \mathrm{SM}'^{-1}.$$

Thus, the file can be reconstructed. The calculations for reconstruction require m additions and m multiplications per byte in the original file.

An important prerequisite for this coding is a set of n vectors of dimension m, such that any m of them are linearly independent. One method of constructing this is the use of Vandermonde matrix [18].

Also note that the CPF's are constructed using modulo p arithmetic, where $p > 2^l$. This may results in "bytes" in the computed CPF's which cannot be represented by a l bit sequence. This problem can be taken care of as follows. Consider $l = 8$ (i.e., an 8-b byte, common with the current technologies). We use the following method for storing a computed "byte" with value greater than 255 $(2^l - 1)$. The bit sequence representing 255 is used as a special character which means that the current byte has overflown. In this case, the next byte in actual storage stores the overflow $(x + 1 - 2^l$, where x is the value of the byte computed). Note that the smallest prime number greater than 255 is 257. So, the storage overhead associated with storing these overflown bytes, when $l = 8$, is less than one percent, which is negligible.

This coding scheme has also been used for providing data security in replicated databases [3]. Also, in [10], a unix implementation of this coding scheme has been presented. Here, a file dispersal shell (FDS) has been developed which uses Rabin's coding for providing a storage efficient disk mirroring scheme.

IV. Using Coding with Simple Voting

This coding can be used to support replicated data in distributed systems. If the data is to be replicated at N nodes, then instead of storing full file at each node (and increasing the disk storage N fold), we can store replicated data using this coding. With coding, we may need to store only one CPF at each node in which case the disk storage is reduced by a factor of m. We will refer to a system with (m, n) coding and N sites participating in the replication as an (m, n, N) system.

With coding, reading and updating a file becomes more complex. The changes made in the original file have to be incorporated into each CPF. A change made in one or more of the m bytes in a row of FM corresponds to a change in one byte at each CPF. Any change in the file can be incorporated in the CPF's by multiplying these rows of FM by the columns of SM corresponding to the CPF's participating in the operation and changing those specific bytes at the individual CPF's.

With this method of storing replicated data, we need proper read and update mechanisms such that data consistency is maintained and high availability is supported. In this section, we present a protocol called simple voting with coding (SVWC), which is a modification of the simple voting method.

In our (m, n, N) system, we assume that $n \leq N$, i.e., number of CPF's does not exceed the number of nodes. Exactly one CPF is stored at each node. Also, each CPF is stored at at least one node and if $n < N$, some CPF's may be stored at more than one node. The n CPF's generated by the coding are labeled $1 \cdots n$. A set of CPF's are said to be all distinct if no two of them are the same. A version number denoting the number of successful updates made on the CPF at that node is maintained at each node. We assume that the nodes can fail only by aborting the transaction [26]. We consider only the simplest form of voting, in which each node (which has exactly one CPF) is assigned one vote each.

A. Quorum Requirements

Since coding is used for storing replicated data, the quorum requirements of SVWC are different from that of voting method. The quorum for the read operations must include at least m distinct CPF's with the latest updates. The write operation must write sufficient CPF's to enable participating set of certain cardinality to include m distinct CPF's with latest update. Here, we study the bounds on the quorum values.

Definition 1: Minimum Sufficient Quorum (MSQ) for read (write) operations is defined as the smallest number of votes which may allow a read (write) operation to complete successfully. We denote this by $r'(w')$.

Clearly, for a (m, n, N) system, MSQ for read and write operations can be given by

$$r' = \max(m, r)$$
$$w' = \max(m, w)$$

where r and w are quorum sizes for read and write operations for simple voting in a system with N nodes. If a write operation updates at least $n'(n' \geq m)$ distinct CPF's in the system, we then have

$$w' = \max(n', w).$$

Definition 2: Maximum necessary quorum (MNQ) for read (write) operation is defined as the largest number of votes which may be required to successfully complete a read (write) operation. We denote this by $r''(w'')$.

Note that in weighted voting with conventional storage, there was no such concept of MNQ and MSQ since the same number of votes would be minimum sufficient and maximum necessary. For example, a read operation cannot be completed without r votes $(r + w > N)$ since mutual exclusion will not be assured and can always be completed with r votes since it will always include at least one site with the latest update.

The MNQ requirements for read and write operations will determine the minimum resiliency of the system for the read and write operations.

Claim 1: In an (m, n, N) system with SVWC where every write operation updates at least n' distinct CPF's, the MNQ for the read operations is given as $r'' = N - n' + m$.

Proof: We need to prove that this is the minimum number of votes which shall ensure mutual exclusion and shall always include at least m distinct CPF's with the latest update. If $r'' = N - n' + m$ and since $w' \geq n'$, we get

$$r'' + w' \geq N + m$$

which means that a read quorum of cardinality $N - n' + m$ will always intersect with any write quorum. This ensures mutual exclusion. Now, a participating set of cardinality $N - n' + m$ or more excludes at the most $n' - m$ sites. Since the write operation had updated at least n' distinct CPF's, at least m CPF's with latest update will be included in such a participating set.

Alternatively, consider a participating set of cardinality less than $N - n' + m$. In the worst case, the last write had updated exactly n' distinct CPF's, $n' - m + 1$ of those are not present in the participating set, so, the operation cannot complete successfully. □

Claim 2: In an (m, n, N) system with SVWC, where every write operation updates at least n' distinct CPF's, the MNQ for write operations is given as $w'' = \max(w', N - n + n')$.

Proof: w'' must be greater than or equal to w' to ensure mutual exclusion. We know, by our system model, that there are n distinct CPF's present in the system. If the participating set has a cardinality of $N - n + n'$ or more, at the most $n - n'$ sites are excluded. So, at the most $n - n'$ distinct CPF's are excluded from the participating set, implying that at least n' distinct CPF's are present in the participating set.

Alternatively, consider a set with cardinality less than $N - n + n'$. Here, $n - n' + 1$ distinct CPF's may be excluded from the participating set, and, in the worst case, they may all be the only CPF's of their kind in the system. So, less than n' distinct CPF's may be present in the participating set. □

B. SVWC

We have seen the bounds on the quorums in the preceding subsection. For a particular operation, the number of the votes in the quorum can be between MSQ and MNQ based on the responses for the request that it receives from other sites. The site initiating a read (write) operation sends read (write) request to all the other sites in the system. The sites reply with their version numbers and the CPF numbers.

The read operation proceeds as follows. The site initiating the operation (initiator) first collects at least r' votes. This ensures mutual exclusion with write operations and necessarily includes at least one site with the last update, so the initiator can determine the latest version number. It then checks if there are at least m distinct CPF's with the latest update. If so, it can read from any m distinct CPF's with latest version number. Else, the quorum is not complete and it waits for responses till it has m distinct CPF's with latest version number. By definition, r'' votes ensure that m distinct CPF's with latest update are included and so the quorum will necessarily be complete.

The write operation is also similar. The initiator first collects at least w' votes to ensure mutual exclusion. Presence of n' distinct CPF's is necessary for the quorum to be complete. It collects votes till n' distinct CPF's are included. A total of w'' votes ensure this and necessarily completes the quorum.

Note that in a read operation, the initiator has to compute the file from the m CPF's it has read. This requires $2m$ operations per byte of the original file. Similarly, if a write operation has to update a sequence of k bytes in the original file, then it requires $(\lceil (k/m) \rceil + 1)mn$ operations in all.

C. Resiliency

Resiliency of a system means the maximum number of failures that can be tolerated while keeping a particular service or operation available. In our system, $N - r''$ and $N - w''$ are, respectively, the resiliency offered by the read and write operations. In voting systems, typically the read resiliency can be improved at the expense of write resiliency and vice-versa. However, the sum of the read and write resiliency remains unchanged and hence can be used as a measure of the effectiveness of the scheme with respect to the fault-tolerance it offers. Therefore, we use this as a measure of the fault-tolerance of SVWC. This value for voting algorithm is $N - 1$. By the claims 1 and 2 we have, assuming $w' \le N - n + n'$

$$2 * N - (r'' + w'') = n - m.$$

This value therefore, does not depend upon n' as long as $w' \le N - n + n'$. By varying n', the values of r'' and w'' can be changed. That is, a higher n' gives lower write resiliency and higher read resiliency, whereas decreasing n' increases write resiliency and reduces read resiliency. The write resiliency has an upper bound of $N - w'$, since w'' has to at least w'. Decreasing n' below $w' + n - N$ does not decrease w'', whereas r'' increases. So, $r + w''$ increases if n' is less than $w' + n - N$.

Moreover, it can be seen that highest overall resiliency (lowest value of $2 * N - (r'' + w'') = N - m$) is achieved when $n = N$, that is, when all the sites have distinct CPF's. Clearly, if all the CPF's are distinct, then lesser number of votes would be required by read and write operations to ensure m distinct CPF's with the latest update and n' distinct CPF's respectively. As n decreases, the resiliency of the system decreases. It can be concluded that to achieve maximum fault-tolerance from the system, one should have a system with $n = N$.

However, there are reasons why a lower n may be preferred. We believe that a replica control protocol should be capable of accommodating changes in the number of sites participating in replication. Consider the case when the number of sites participating in the replication may increase. Now, adding a new vector, such that it is linearly independent with any set of $m-1$ of the n vectors participating in the coding being used in the system, can be a very difficult task. Whereas, changing the system from (m, n, N) to $(m, n, N+1)$ may be an easier task, involving only change in the quorum requirements [14], [20]. The system will then be operating with lower n as compared to N. Moreover, the computational cost of determining the various CPF's increases as the number of distinct CPF's increases. Thus, we will prefer a system with a comparatively lower n as compared to N.

Example: Consider an (3, 10, 12) system. The value of $r'' + w''$ for SVWC is 17, provided $n' \ge w' - 2$. This value for a 12 node system with conventional voting is 13. If we choose $r = 4, w = 9$ and $n' = 10$, we have $r' = 4$, $w' = 9$, $r'' = 5$ and $w'' = 12$. The resiliency for write operations may be improved by decreasing n', for $n' = 7$, we have $r'' = 8$ and $w'' = 9$. The resiliency for write operations cannot be increased any further since further decrease in n' increases r'' without decreasing w''.

D. Dynamic Redistribution of CPF's

We saw in previous subsections that to ensure the presence of m distinct CPF's in the participating set with certain number of votes, the write operations are required to update at least n' distinct CPF's, resulting in a high MNQ for write operations. These conditions apply to a system where CPF's are statically distributed among the nodes.

We can achieve higher resiliency with lower n if we allow dynamic change in the distribution of CPF's on the nodes in the system. With this approach, if during a write operation involving k sites ($k \ge n'$) only $l (l < n')$ distinct CPF's are present in the participating set, then the write operation completely rewrites some duplicate CPF's in the participating set such that, at the end of the write operation, the participating set consists of at least n' distinct CPF's.

For example, suppose the participating set consists of exactly n' sites, containing only $n' - 1$ distinct CPF's, such that two sites have the ith CPF while jth CPF is not present at all. The write operation may replace an ith CPF by an updated jth CPF, while all the other CPF's may be updated at their respective sites. The write has, now been done on n' distinct CPF's, without the write set initially including n' distinct CPF's.

By this policy, a write operation "redistributes" the CPF's in the system. By the method the CPF's are replaced by other, at least one copy of each of the n distinct CPF's is always present in the system.

The SVWC changes as follows. The read algorithm remains unchanged. The write algorithm no longer needs to include n' sites with distinct CPF's, it simply needs to include n' sites. The MSQ requirements and claim 1 remain the same. To give maximal resiliency we have

$$n' = \min(w', n).$$

This is because the write quorum will involve at least w' sites. If $w' \geq n$ and if the write operation updates at least w' CPF's, then the MNQ requirements for read can be reduced. Taking $n' > w'$ would increase the MNQ requirement for write operations. But n' has to be less than or equal to n anyway. With this value of n', the MNQ for read (from Claim 1) is

$$\begin{aligned} r'' &= N - w' + m, \qquad \text{if } n \geq w' \\ &= N - n + m, \qquad \text{otherwise.} \end{aligned}$$

Claim 2 no longer holds since the write operation does not need to include n' distinct CPF's. Since $n' \leq w'$, the presence of w' sites would be enough to ensure n' CPF's in the quorum. However, if the write operation needs to redistribute the CPF's, and was otherwise not writing the entire data item, it may need to perform a read operation to be able to write a full CPF at the sites where the CPF is being replaced. The MNQ for write is, therefore

$$w'' = \max(w', r'').$$

We see that if $n \geq w'$, then $r'' + w' = N + m$. If r and w have been so chosen that $w' \geq r''$, then $2 * N - (r'' + w'') = N - m$. So, for all choices of n from w' to N, the sum of the resiliency of read and write operations is $N - m$. This implies that with the dynamic redistribution of CPF's, we can achieve better fault-tolerance, even with lower n.

Note that the restrictions for achieving $N - m$ as the sum for read and write resiliency are $n \geq w' \geq r''$. $w' \geq r''$ is not a serious restriction since, in voting systems the write quorum is generally greater than the read quorum anyway. The only restriction therefore is that n should be at least w'.

Example: Consider a (3, 10, 12) system again. If we take $r = 4$ and $w = 9$ again, we get $n' = 9$. This gives $r' = 4, r'' = 9, r'' = 6$ and $w'' = 9$, giving $r'' + w'' = 15$, a clear improvement over 17 necessary without dynamic redistribution. Note that this could have been achieved for any value of n from 9 to 12. If $n = 8, n'$ becomes 8 then r'' changes to 7. $r'' + w''$ is then 16.

V. Performance

In the previous sections we have seen how coding can be used to store replicated data. We also saw SVWC, a new scheme for managing replication when coding is used to store replicated data. Clearly, if coding is used, the storage requirements can be reduced by a factor of m. However, since the quorum requirements have changed, the availability offered is lowered. We, therefore, study the reductions in the storage space to store the data with a given amount of availability. In this section only, we also study the changes in the communication overheads with the proposed scheme.

A. Availability

Availability is the most important performance metric of any voting system. The definition of availability that we will use is as follows. Availability of read (write) operations of any system is the probability, in the steady state, of a read (write) quorum being available in the system. To evaluate the availability, we make the following assumptions about our system. We assume that each site is operational at any time with a probability p (called availability of a node), independent of any other site. Further, we assume that no network partitions take place. This assumption is required to separate our analysis from numerous network topologies that may exist in a distributed system. This assumption has already been used for analysis in [5], [8], [15], [19]–[21]. Also, its has been shown in [17] that the results on the availability are qualitatively the same when the analysis is done with or without considering the network partitions.

For simplicity, we assume that operations take place only when MNQ is met. Note that the actual availability of SVWC is more than what we evaluate here since operations may be completed even with quorum values less than MNQ. In our (m, n, N) system, we assume that $n \geq N + m/2$ and dynamic redistribution is used. One possible quorum assignment for SVWC gives

$$w = \left\lceil \frac{N+m}{2} \right\rceil, \quad r'' = \left\lfloor \frac{N+m}{2} \right\rfloor.$$

A similar assignment for simple voting for a system of N nodes is

$$w = \left\lceil \frac{N+1}{2} \right\rceil, \quad r = \left\lfloor \frac{N+1}{2} \right\rfloor.$$

We compare the availability for write operations using these vote assignments.

The availability of write operations for conventional voting is

$$P[X \geq w] = \sum_{i=w}^{N} \binom{N}{i} p^i (1-p)^{N-i}.$$

where, X is the number of operational sites at any time. The availability of write operations for the (m, n, N) system is

$$P[X \geq w] = \sum_{i=w}^{N} \binom{N}{i} p^i (1-p)^{N-i}.$$

The storage factor (denoted by l) of a (m, n, N) system is N/m. A storage factor of l means that the overall storage in the system is l times the size of the file being replicated. The

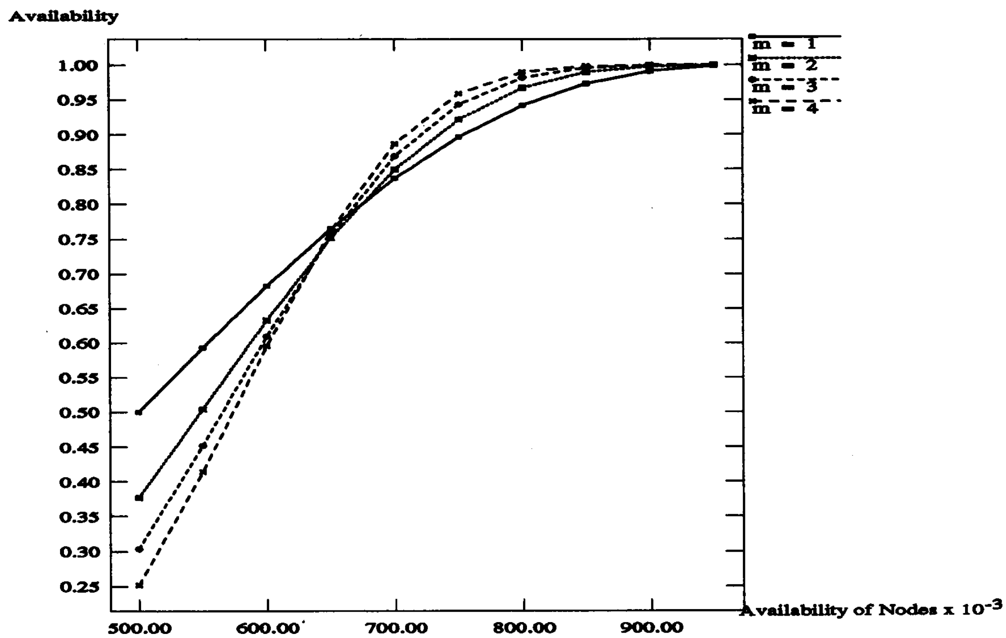

Fig. 1. Availability of system versus availability of nodes (1 = 5).

storage factor, therefore, represents the storage overhead in the system. In the discussion that follows, system with $m = 1$ means simple voting (i.e., without coding).

Fig. 1 shows how the availability of $(m, n, l * m)$ system varies with the availability of nodes (p) for the same storage factor (fixed at $l = 5$). We can see that, except when the nodes have very low availability $(p < .70)$, the availability increases as m increases. The relatively poor performance of voting with coding at higher m when $p < .70$ can be explained as follows. In coding with parameter m, the expression of availability excludes terms $(N/i)p^i(1-p)^i$, from $i = \lceil (N + 1/2) \rceil$ to $\lceil (N + m/2) \rceil - 1$ as compared to the availability without coding. In the binary expansion of $(p + (1-p))^N$, the central terms become significant if p is close to 0.50. Hence, the availability offered reduces. However, for p close to 1, only the end terms in the binary expansion are significant, hence, the availability offered (with a given N) is high even with a higher value of m.

In Fig. 2, we show the storage factor required by $(m, n, l * m)$ system to achieve desired availabilities, (p is fixed at .90). For high availabilities, as m increases, the storage factor required decreases. The reduction in the storage space is not achieved when the availability required is low. This is because when m is close to N, the probability of finding m distinct CPF's in the participating set is low.

In Fig. 3, the storage factor required to achieve availability of .999 is shown at different values of p. Clearly, the storage factor required decreases with increasing m. Moreover, with $m = 3$ or $m = 4$, the reduction in storage space is generally better than a factor of two. So, our scheme can definitely offer a much greater reduction in storage space than the fragmentation based approach.

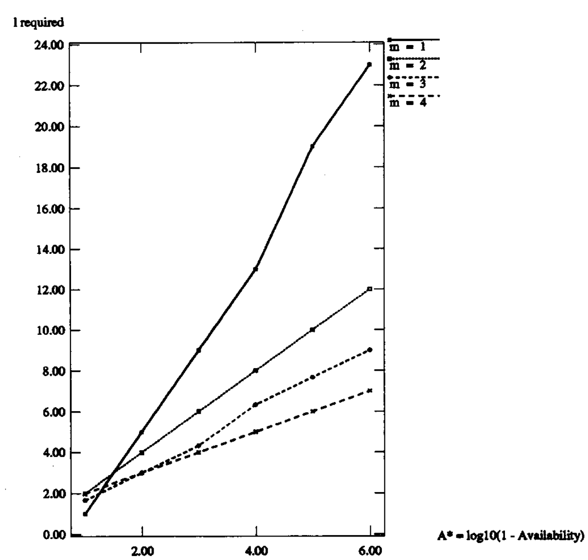

Fig. 2. l required versus availability, $p = .90$.

B. Communication Overheads

To study the communication overheads associated with the proposed scheme, we use *bandwidth factor* as the metric. The bandwidth factor is the mean sum of the message sizes for an operation. It is measured as a multiple of the size of the original file on which read or write operations are being done.

The communication overhead corresponding to the broadcast of the read or write request is clearly proportional to the number of sites participating in the replication. The number

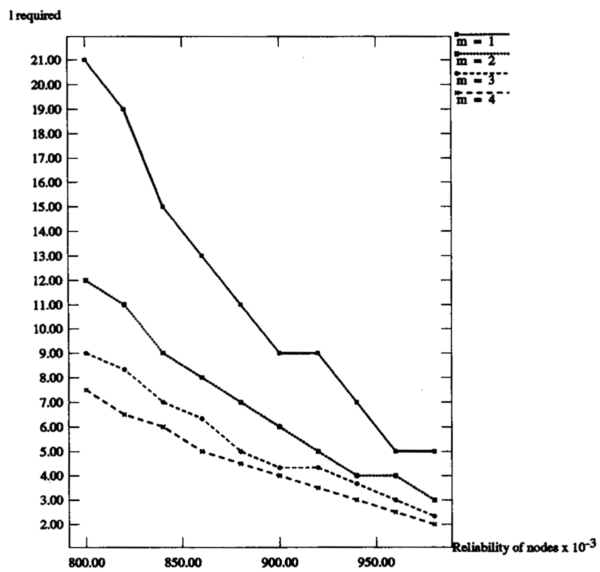

Fig. 3. l required at different p, availability = .999.

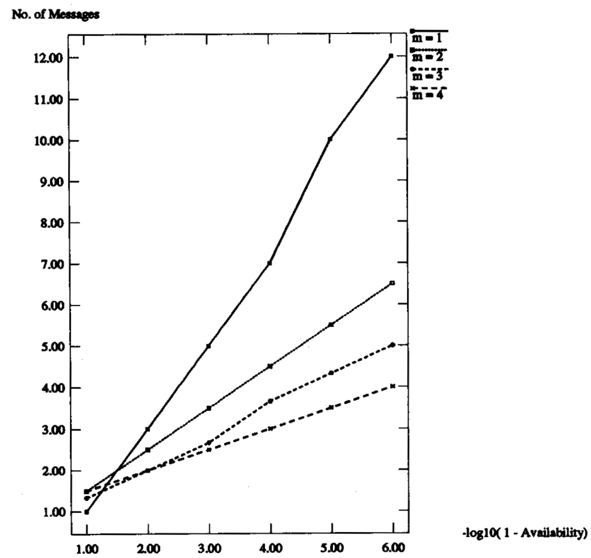

Fig. 4. Bandwidth factor required by write operations versus availability (p = .90).

of messages required for writing the replicated data after gathering quorum is $\lceil N + m/2 \rceil - 1$, assuming that the site initiating the operation is one of the sites written into. The bandwidth factor is $(1/m) * (\lceil N + m/2 \rceil - 1)$.

In Fig. 4, we compare the bandwidth factor required for write operations for different values of m, when the availability required is varied ($p = .90$). The bandwidth requirement for write operations decreases considerably with the use of coding and with increasing m. Note that we are comparing the systems with different values of m when they provide the same availability. Thus, N increases with m. However, even with increased N (and hence increased write quorum $\lceil N + m/2 \rceil$), the bandwidth requirements decreases significantly with increase in m. This is because the N required to maintain the same availability increases only marginally with increase in m and the bandwidth factor required for each message is $1/m$.

For a read operation, data from m sites are read. Assuming that no failure (or repair) has occurred since the last write in the system and the read operation is equally likely to arrive at all the operational nodes, the probability that the site initiating the read request does not have the latest update is

$$\frac{\sum_{i=w''}^{N} \binom{N}{i} p^i (1-p)^{N-i} \frac{i-w''}{i}}{\text{Availability for write operations}}.$$

Note that if i sites ($i \geq w''$), participated in a write operation and if only w'' of them were updated, then the probability of one of these i sites not having the latest update is $i - w''/i$. The average number of messages required is $m - 1$ plus the probability that the site initiating the operation does not have the latest update (shown above). The bandwidth factor required is the average number of messages required divided by m.

In Fig. 5, we study the bandwidth factor required for read operations, at different values of m, when availability required is varied ($p = .90$). For read operations the bandwidth requirement increases as m increases.

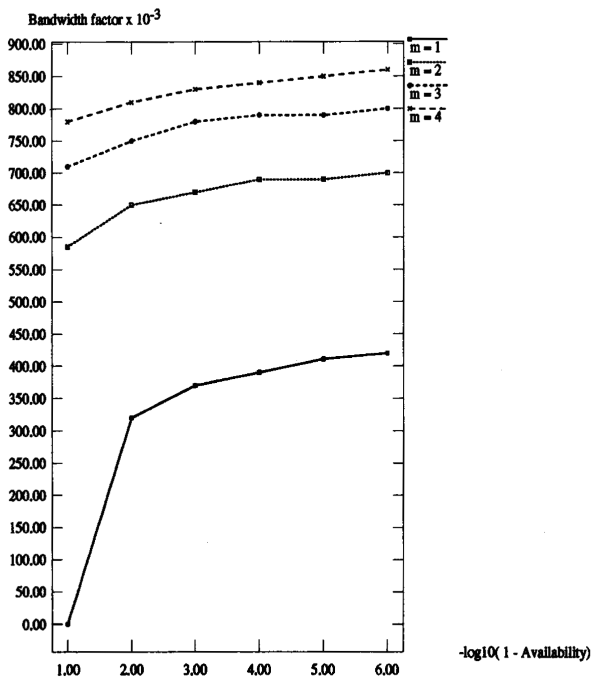

Fig. 5. Bandwidth factor required by read operations versus availability (p = .90).

From the analysis shown in this section, it is clear that the proposed scheme SVWC gives better availability for same storage factor. Alternatively, the storage requirements for achieving certain desired availability decreases with the the proposed method. Note that the reduction in the storage requirements achieved is much more than that from the fragmentation based approach and Voting with Witnesses. The only disadvantage is the increase in the communication overhead for read operations.

VI. Dynamic Voting with Coding

We have seen how the simple voting protocol can be modified to manage coding based replication. However, other replica control protocols can also be modified to manage such replications. In this section we present dynamic voting with coding, a new scheme for managing coding based replication. This scheme is derived from the dynamic voting algorithm suggested by Jajodia and Mutchler [15]–[17]. The dynamic voting algorithm algorithm of Jajodia and Mutchler is, in turn, a modification of a previous replica control protocol suggested by Davcev and Burkhard [11].

A. Dynamic Voting Protocols

In the dynamic voting protocol, the quorum requirements for read and write operations have been modified to enhance availability in face of network partitions. In this algorithm, any operation needs to collect a majority of the sites which participated in the last update rather than a majority of all the sites in the system. All sites participating in an update operation record the total number of sites participating in that operation in a variable called update site cardinality (SC). All these sites then have the highest version in the system. For the next update or read operation, at least a majority of the SC($\lceil \mathrm{SC} + 1/2 \rceil$) sites with the latest version number are required. In this protocol all the sites available during a write operation are updated, rather than updating just the sites required for majority of SC.

Dynamic voting with coding (DVWC) operates in a similar fashion. The only modification required is that any operation needs at least m distinct CPF's with the latest version number besides a majority of SC. For the simplicity of our discussion, we assume that $n = N$. Now, the quorum requirements for any operation in the system is $\max(m, \lceil \mathrm{SC}+1/2 \rceil)$. With this, the operation of DVWC remains the same as that of dynamic voting as long as SC for any update is at least $2m - 1$.

B. Availability Analysis

We evaluate the availability offered by this protocol to compare the storage space requirements with those of dynamic voting. Again, we assume that network partitions do not take place. To make our analysis feasible through stochastic models, we assume that the a failure (recovery) arrives at any operational (unoperational) site with a rate $\lambda(\mu)$. We also assume that updates are much frequent than failures and recoveries so that whenever any update or failure occurs, an update arrives immediately with that topology. This is called *frequent update assumption.* This assumption was used by Jajodia and Mutchler for the original analysis of dynamic voting [17]. This assumption may be quite reasonable for certain applications, else, it may be satisfied by the use of frequent polling [17].

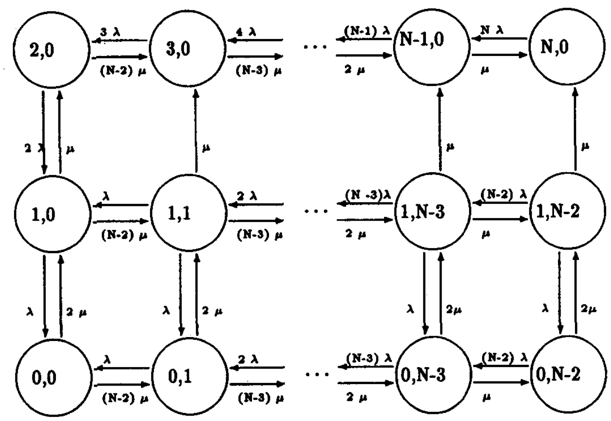

Fig. 6. Markov chain representation of dynamic voting.

We use a Markov model to compute the availability of these schemes. Under the above assumptions, the operation of dynamic voting is as follows. Whenever SC is 3 or more, the system is always available since if a failure arrives, still SC − 1 sites with the latest update are available, this is a majority of SC. Because of the frequent update assumption, another update occurs in the system with SC one less than the SC for previous update. If SC is 2 and both the sites with the latest update are up, then also the system is available. However, if SC is 2 and a failure occurs, then no operations will be possible until both these sites with the latest update are able to recover.

A Markov model for dynamic voting is shown in Fig. 6. States are denoted by (X, Y) where X is the number of operational sites with the latest update and Y is the number of operational sites without the latest update. Note that if $X \geq 2$, then $\mathrm{SC} = X$ and $Y = 0$. The system will be available in this case. If $X < 2$, then $\mathrm{SC} = 2$ and the system is unavailable.

DVWC operates similarly with the parameter m replacing 2. If $\mathrm{SC} > m$, then the system will definitely be available since a majority of sites with the latest update as well as m sites with the latest update are available. If SC is m and then a failure strikes, then the system will be unavailable until all these m sites with the latest update recover. Markov model for DVWC is shown in Fig. 7.

C. Results

These Markov chains were solved by giving a generalized stochastic Petri net (GSPN) description of these and using the stochastic Petri net package (SPNP) [6]. An interesting aspect of this analysis is that with the frequent update assumption, the operation (in terms of quorum availability) of DVWC with $m = 2$ is same as that of Dynamic Voting. This is because operation of dynamic voting is a special case of DVWC with $m = 2$. A reduction in storage space by a factor of 2 is, therefore, achieved straight away. If the failures (recoveries) occur at any operational (unoperational) sites with rate $\lambda(\mu)$, then in steady state, the probability of any site being operation is $\mu/(\mu + \lambda)$. We denote this by p.

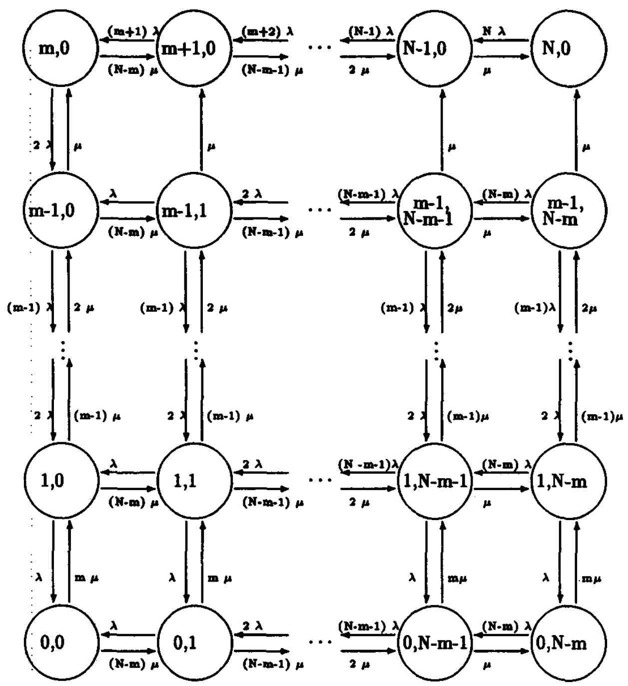

Fig. 7. Markov chain representation of DVWC.

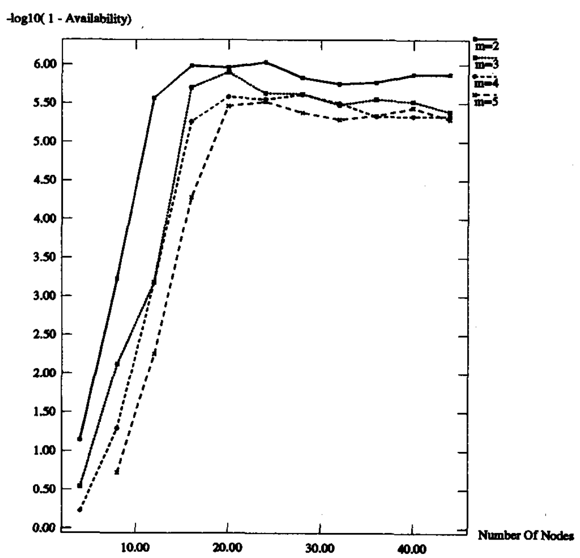

Fig. 8. Availability offered versus number of nodes, (p = .80).

In Fig. 8, we compare the availability offered by the dynamic voting with coding at different N, keeping p fixed at .80. We compare the availability offered for $m = 2 \cdots 5$. Note that the results for simple dynamic voting are same as that of DVWC and $m = 2$.

From this graph we see that with dynamic voting (with or without coding), increasing the number of nodes does not indefinitely increase the availability (unlike simple voting). This may be explained as follows. While increasing the number of nodes decreases the probability of the system entering the state where SC is m and not all the nodes with the highest version number are up, the expected time that the system (after having reached such a state) will take to be able to offer availability again also increases with N. Hence, availability cannot increase indefinitely with increasing N.

Another point to note is that the maximum availability that can be offered decreases with increase in m. This is because the probability of reaching a state where SC is m and not all these sites are up increases with increase in m.

However, for the realistic values of availability, the storage space required does decrease with increasing m. With $m = 2$, the storage requirements decrease to half as compared to the full file replication. In Fig. 9, we compare the storage space requirements for getting the availability of .999, for $m = 2 \cdots 5$. Clearly, significant reductions in the storage space requirements can be achieved with higher values of m.

VII. Coding with Other Voting Schemes

In this section we discuss how two of the extensions to the simple voting, i.e., weighted voting and the approach based on coteries can also be modified to maintain correctness when coding is used to store replicated data.

A. Weighted Voting

Weighted voting is a generalization of simple voting in which a site can be assigned any number of votes. In [13],

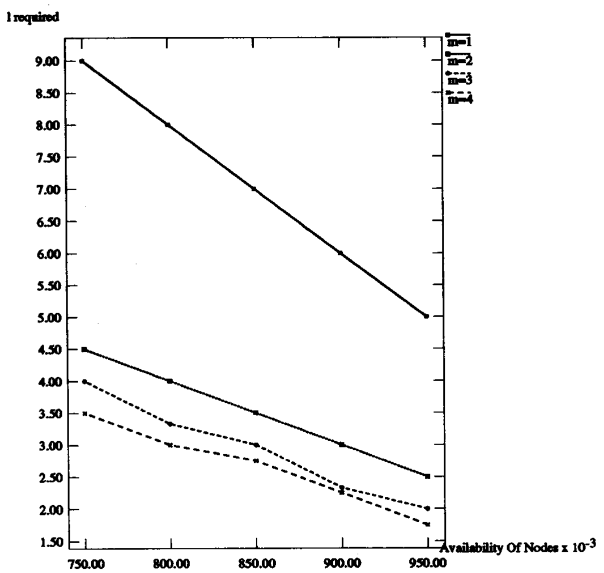

Fig. 9. l required versus p, (availability = .999).

it has been shown that for a homogeneous system with odd number of sites, assigning 1 vote to each site gives maximum availability. If the number of sites is even, then giving one of the nodes an extra vote helps in breaking ties (and improves availability). In heterogeneous systems, however, more reliable nodes can be given higher number of votes and this may help improve availability. Also, giving higher number of votes to sites having larger number of read and write requests coming to them may also help improve performance of read and write operations.

When coding is used for storing replicated data, sites may be given different number of votes. However, for the MNQ requirements shown in Sections IV to hold, a site with k votes needs to keep k CPF's if $(k \leq m)$ and m CPF's if $k > m$. This is because in the computations for MNQ, we assume that intersection of read and write quorum at m votes will imply the intersection at m CPF's also.

Clearly, giving higher weights to some of the sites increases the overall storage requirement in the system. Note that this wasn't true with the full file replication since one full copy of the file is stored at each of the sites irrespective of the number of votes assigned to that site. However, the storage site in weighted voting with coding never exceeds the storage requirement of full file replication, since at the most m CPF's (i.e. one full copy) is stored at each site.

The main purpose of giving unequal votes to nodes may be that by giving higher number of votes to nodes with higher load, performance of read and write operations can be improved. A node with higher number of votes will have higher number of CPF's, so if the CPF's at that site have the highest version number, the communication cost for read operations will be reduced. Similarly during the write operations, not many other sites may be required to be included in the quorum, again the communication costs will be reduced.

B. *The Method of Coteries*

The approach based on coteries in [13] is a generalization of the weighted voting method. Consider the set of nodes of the system. A coterie is defined to be a set of subsets of this set of nodes. These member subsets are such that each intersects every other at at least one node and no such subset is a proper superset of any other member subset. The quorum requirement with this approach is that a set of nodes forms a quorum only if it includes a member subset of the coterie. Clearly, since these member subsets intersect, mutual exclusion of read-write and write-write operations is ensured.

This approach may be modified to work with the coding as follows. Again, we store one CPF at each of the nodes. We define *m-coterie*, which is similar to coterie except that each member subset must intersect with every other member subset at at least m sites. For simplicity of discussion, we assume that number of distinct CPF's in system is at least equal to the cardinality of the largest member subset in the coterie.

The quorum requirements will now be as follows. A write quorum must include one such member subset of the coterie. The write operation will use dynamic redistribution so that after the write, all the nodes in the member subset have distinct CPF's. MSQ for read operations is a set of nodes such that they intersect every member subset of the coterie at at least one node. This ensures mutual exclusion and the highest version number existing in the system can be known. If there are m distinct CPF's with the highest version number, then the read operation can be completed. The MNQ for read quorum is same as the quorum requirement for the write operation.

VIII. Discussion

Data replication is often used to enhance the availability and performance in Distributed Systems. This replication of data incurs a high storage overhead. In this paper, we have presented schemes which significantly reduce the storage requirements for maintaining the data with a given availability. These schemes use the coding suggested by Rabin [2], [3] to store replicated data.

In the proposed schemes, only coded parts of files are stored at individual nodes rather than storing full files at each node, as in the conventional schemes. m of these parts are required to reconstruct the file. To use this coding for storing replicated data, we need proper read and write algorithms.

The first scheme that we presented (SVWC), is a modification of the simple voting algorithm. The algorithm and the quorum requirements are modified to manage such coding based replication. We initially considered the case when the distribution of these parts is static. Then, we extended this to the case when the parts are dynamically redistributed during the write operations. This dynamic redistribution improved the resiliency. The second scheme that we presented (DVWC) is similarly derived from dynamic voting algorithm of Jajodia and Mutchler.

We evaluated the availability offered by both these schemes. The results show that the storage space required to achieve certain availability reduces significantly with the proposed schemes. Alternatively, better availability can be achieved by using the same storage space. We also briefly discuss how some other extensions to the voting algorithms can also be modified to derive storage efficient schemes for maintaining replicated data.

Our method achieves much better reductions in the storage requirements than the other approaches previously proposed for this purpose. The maximum reduction that the fragmentation based scheme can provide is by a factor of 2. Our schemes, with the values of m 3 and above can provide much better reduction in the storage requirements. Also, our schemes do not suffer from the disadvantages that the fragmentation based scheme suffers from. The fragmentation based scheme is incapable of accommodating changes in the number of sites participating in the replication. This is because in this scheme, the number of sites participating in the replication is the same as the number of fragments into which the file is divided. In the proposed scheme, adding (or deleting) one site from the system is simple, the system changes from (m, n, N) to $(m, n, N+1)$ (or, $(m, n, N-1)$), requiring only a change in the quorum requirements [14], [20].

In the fragmentation based scheme, if the size of the file is changed significantly, the fragments will need to created again, incurring significant overheads. The proposed scheme does not have this disadvantage. Incremental updates can be made on each of the CPF's to accommodate any change made in the original file. Also, note that both the proposed scheme and the fragmentation breplication schemes. The proposed scheme, however, does not add any complexity in the operation of the voting protocol. The fragmentation based scheme, on the other hand, requires a complicated propagation scheme to maintain sufficient resiliency. We believe that such a scheme would be hard to implement. Further, this propagation based scheme requires significant storage and communication overheads, the exact values of which are hard to assess.

The use of Rabin's coding in replicated databases also gives a high degree of data security [3]. Security of data in a replicated distributed system may be a real concern, because an unauthorized user may break into the data by simply breaking into the protection scheme at any of the N nodes where the data has been replicated. Also, during a read or write operation, the files may be read by an adversary if the communication links through which it is being transferred are insecure, as it happens with most of the existing networking technologies.

Some efforts for enhancing security in distributed systems are [00], [00], and [00]. In [00], Shamir suggests a method in which the information of the file is distributed over N files, each of the size of the original file, such that any m of then suffice to reconstruct the file. (N and m are parameters, s.t. $m \leq N$). While this method gives sufficient security, this is not storage efficient as each of the N files are as large as the original file. The approaches suggested in [00], [00] also treat the issue of data replication separate from that of data security.

Use of Rabin's coding for storing replicated data overcomes these problems and gives a high security to the replicated data. An unauthorized user has to break into at least m nodes and also has to decode the coding being used in order to read a file. Also, read and write operations require only individual CPF's to be transferred on a communication link at a time and not the entire file, which provides security against network tapping also.

Note that our method requires $2m$ operations per byte of the original file for the read operation and $2n$ operations per byte for write operations. (m and n are parameters of the coding) Obviously, if the data is read and updated frequently, this may be a considerable overhead. However, this cost may be justified by the very significant reductions in the storage space requirements. Also note that our method provides data security as well. Any other scheme used for maintaining data security will also incur similar overhead in coding and decoding files.

Overall, we believe that the proposed scheme can be very useful in supporting replicated data in a storage efficient manner, in addition to providing data security.

ACKNOWLEDGMENT

The analysis presented in Section VI of this paper was done with the help of SPNP [6], a software for solving general stochastic Petri nets. This tool has been developed at Duke University. The authors are grateful to D. Liang for having introduced them to this package. The authors also thank W. Burkhard for suggesting [18] for constructing n m-dimension linear vectors such that any m of them are linearly independent. The comments from the referees greatly helped in improving the presentation of this paper.

REFERENCES

[1] D. Agrawal and A. El Abbadi, "Reducing storage for quorum consensus algorithms," in *Proc. Very Large Databases Conf.*, 1988, pp. 419–430.
[2] ——, "Storage efficient replicated databases," *IEEE Trans. Data, Knowl. Eng.*, vol. 2, pp. 342–351, Sept. 1990.
[3] ——, "Integrating security with fault tolerance in distributed databases," in *Comput. J.*, vol. 33, no. 2, pp. 71–78, Feb. 1990.
[4] G. Agrawal and P. Jalote, "An efficient protocol for voting in dist. systems," in *Proc. 12th Int. Conf. Dist. Comput. Syst.*, June 1992.
[5] M. Ahamad and M. H. Amnar, "Performance characterization of quorum consensus algorithms for replicated data," *IEEE Trans. Software Eng.*, vol. 15, pp. 492–496, Apr. 1989.
[6] G. Ciardo and J. K. Muppala, *Manual for SPNP Package version 3.0*, Duke University, Durham, NC, July 1990.
[7] D. Barbara and H. G. Molina, "Vulnerability of voting mechanisms," *ACM Trans. Comput. Syst.*, vol. 4, no. 3, pp. 187–213, Aug. 1986.
[8] ——, "Reliability of voting mechanisms," *IEEE Trans. Comput.*, vol. 36, pp. 1197–1208, Oct. 1987.
[9] P. Bernstein and N. Goodman, "An algorithm for concurrency control and recovery in replicated distributed databases," *ACM Trans. Database Syst.* vol. 9, no. 4, pp. 596–615, 1984.
[10] W. A. Burkhard and P. D. Stojadinovic "Storage efficient reliable files," in *Proc. 1992 Winter USENIX Conf.*, 1992, pp. 69–77.
[11] D. Davcev and W. A. Burkhard, "Consistency and recovery control for replicated files," in *Proc. 10th ACM Symp. OS Principles*, 1985.
[12] S. Davidson and H. Molina, "Consistency in partitioned networks," *ACM Comput. Surveys*, vol. 17, no. 3, pp. 341–370, 1985.
[13] H. G. Molina and D. Barbara, "How to assign votes in a distributed system," *J. ACM*, vol. 32, no. 4, pp. 841–860, Oct. 1985.
[14] D. K. Gifford, "Weighted voting," in *Proc. 7th ACM Symp. OS Principles*, 1979, pp. 150–162.
[15] S. Jajodia and D. Mutchler, "Dynamic voting," in *Proc. ACM SIGMOD*, 1987, pp. 227–238.
[16] S. Jajodia and D. Mutchler, "Enhancements to the voting algorithm," in *Proc. 13th Int. Conf. Very Large Databases*, 1987, pp. 399–406.

[17] S. Jajodia and D. Mutchler, "Dynamic voting algorithms," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 231–280, June 1990.

[18] L. Mirsky, *An Introduction to Linear Algebra.* New York: Dover, 1982.

[19] J. F. Paris, "Voting with witnesses: a consistency scheme for replicated files," in *Proc. 6th IEEE Int. Conf. on Distrib. Comput. Syst.*, 1986, pp. 606–612.

[20] ——, "Voting with a variable number of copies," in *Proc. 16th IEEE Fault-Tolerant Comput. Symp.*, 1986, pp. 50–55.

[21] ——, "Voting with bystanders," in *Proc. 9th Int. Conf. Distrib. Comput. Syst.*, 1989, pp. 394–401.

[22] C. Pu, J. D. Noe, and A. Proudfoot, "Regeneration of replicated objects: A technique and its eden implementation," in *Proc. 3rd IEEE Int. Conf. Data Eng.*, 1987, pp. 175–187.

[23] M. O. Rabin, "Efficient dispersal of information for security, load balancing and fault-tolerance," Harvard University, Cambridge, MA, TR-02-87, Apr. 1987.

[24] ——, "Efficient dispersal of information for security, load balancing and fault-tolerance," *J. ACM*, vol. 36, no. 2, pp. 335–348, 1989.

[25] R. van Renesse and A. S. Tannenbaum, "Voting with ghosts," in *Proc. 8th IEEE Int. Conf. Distrib. Comput. Syst.*, 1988, pp. 456–461.

[26] F. Schneider and Schlichting, "Fail-stop processors: An approach to designing fault-tolerant distributed systems," *ACM Trans. Comput. Sys.*, vol. 1, no. 3, pp. 222–238, Aug. 1983.

Gagan Agrawal was born in India in 1970. He received the Bachelor of Technology degree in computer science and engineering from the Indian Institute of Technology, Kanpur, in 1991.

He is currently a Ph.D. student at the Department of Computer Science, University of Maryland, Kanpur. His research interests include parallel computing, compilers and runtime support for distributed memory parallel machines and fault-tolerance.

Pankaj Jalote (M'91–SM'92) received the B.Tech. degree from the Indian Institute of Technology, Kanpur in 1980, the M.S. degree from Pennsylvania State University, University Park in 1982, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1985.

From 1985 to 1989 he was an Assistant Professor in the Department of Computer Science at the University of Maryland College Park. Since 1989 he has been at the Department of Computer Science and Engineering at the Indian Institute of Technology, Kanpur, where he is currently an Associate Professor. His research interests are software engineering, fault tolerant computing, and distributed systems. He is the author of the book, *An Integrated Approach to Software Engineering* (New York: Springer-Verlag, 1991).