Distributed and Parallel Databases, , 1–25 (1999)

Diamond Quorum Consensus for High Capacity and Efficiency in a Replicated Database System

ADA WAI-CHEE FU AND YAT SHEUNG WONG AND MAN HON WONG

adafu@cse.cuhk.edu.hk, mhwong@cse.cuhk.edu.hk

Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong

Abstract.
Many quorum consensus protocols have been proposed for the management of replicated data in a distributed environment. The advantages of a replicated database system over a non-replicated one include high availability and low response time. We note further that the multiple sites can act as multiple agents so that at any time, multiple requests can be handled in parallel. This feature leads to the desirable consequence of high workload capacity. In this paper, we define a new metric of *read-capacity* for this feature. We propose a new protocol called *diamond quorum consensus* which has two major properties that are superior to the previous protocols of majority, tree, grid, and hierarchical quorum consensus: (1) it has the highest read-capacity, (2) it has the smallest *optimal* read quorum size of 2. We show that these two features are achievable without jeopardizing the availability. The small quorum size is a significant feature because it relates to the messaging cost. Few previous work on quorum consensus has discussed the handling of partition failure, which in many cases will depend on the quorum consensus protocol, we show how we can use the *generalized virtual partition protocol* to handle partition failure in the case of diamond quorum consensus.

Keywords: mutual exclusion, quorum consensus, distributed systems, replicated database systems, fault-tolerant computing

1. Introduction

A replicated database system is built in a distributed environment with multiple sites, in which copies of data are stored at multiple sites. The main motivations of a replicated database are to improve the reliability and performance. By storing data at multiple sites, the database system can continue to operate even if some sites have failed. Also, multiple sites make it possible to support multiple concurrent operations at different sets of sites.

A major problem with replicated data is that we need to ensure data consistency in spite of concurrent operations. A reliable concurrency control protocol is necessary to *synchronize* the user transactions in order to maintain data validity. For example, two write operations from two different user transactions must not be allowed to simultaneously update different copies of a data object. In order to achieve this kind of synchronization among multiple copies, additional communication and processing costs are incurred. We have a problem of how to do the synchronization in such a way that the cost can be minimized. One well-studied approach for this management of replicated data is to use certain sets of replicas, called *read* and *write quorums*, for read and write operations, respectively. Any write quorum has at least one copy in common with every read quorum and every write quorum. For example, write quorums could be all sets containing a majority of copies [18]. For better performance, some logical structure is imposed on the network, and the quorums are chosen under the consideration of such structures. Such logical structures include the tree [2, 1, 3, 20] and grid [4, 11] structures. A geometric approach for dealing with logical structures is proposed in [12]. A number of metrics have been used for evaluating such a protocol, they include the following:

1. Availability
2. Quorum size (best case and worst case)
3. Is the algorithm fully distributed?
4. Cost of one node failure in the worst case
5. Message overhead
6. Communication delay

The first metric on availability has been studied extensively but its significance has decreased as systems tend to be more and more reliable, a protocol needs only to have high availabilities under a reasonably reliable environment. Metrics (2) to (4) are listed in [10]. The quorum size is also a well-studied issue because it can be related to the messaging cost. From [10], the $\sqrt{N}$ algorithm in [13] is a *fully distributed* algorithm that achieves the smallest quorum size. The protocol in [16] has a bigger quorum size but remedies the availability and other problems of [13].

The third metric is whether all copies assume equal burden for synchronization, for example, the tree quorum is not fully distributed because the nodes higher in the tree are given more share of the burden in order to achieve a small quorum size. However, we shall argue below that this metric is problematic. The fourth metric is the impact that a single node failure has on the size of a quorum. For example, for the $\sqrt{N}$ algorithm of [13], if one of the sites in a chosen quorum fails, then in the worst case, a new quorum must be formed with a totally different set of $\sqrt{N}$ nodes. The average message overhead of replica control protocols is studied in [17]. Communication delay is studied in [7, 8]. For communication overhead, both the quorum size and the communication protocol are examined [9].

Other than availability, the above metrics try to ensure two important performance criteria of a database system: fast response time and high throughput, even when the system is under heavy workload. However, we argue here that while these metrics are useful, they are not sufficient. Let us look at the second metric of quorum size. A quorum size can be small, but if each request is targeted at the same set of sites, these sites will become a bottleneck and when the system is busy, both the throughput and response time of the system will become poor. The third metric of even workload distribution is relevant, it is set to be a yes/no metric in [10]. An even distribution of workload may not necessarily lead to good performance under heavy workload. For example, the majority quorum consensus has even workload distribution, but at any one time, only one operation can proceed, which means that the performance cannot be enhanced by concurrent operations. Therefore, we would need another metric for the performance under heavy workload. In this paper, we introduce such a metric, which we shall call the **read-capacity**.

Let us consider a replicated database system, it is in a distributed environment with multiple sites, where copies of a data are kept at different sites. For quorum consensus, when a read (write) operation is executed, it must obtain the consensus from a read (write) quorum. A write quorum intersects each other write quorum and each read quorum, therefore at any time only one transaction can be writing a logical data. A read quorum needs only to intersect a write quorum, therefore, as long as two read quorums do not intersect each other, two read operations on the same logical data can be executed concurrently. One may think that the read quorums of two concurrent read operations can intersect each other, but when we measure the throughput or response time, we must consider the amount of work involved in a read operation for each copy. In particular, the messaging to and from each site in a quorum will be a major overhead, and an intersection in read quorums will create a bottleneck. This is why the metric of even workload distribution is important. If two read quorums are disjoint, there will not be any bottleneck for response time and throughput. In view of the above, we introduce a metric here that measures *the maximal number of read operations that can be handled simultaneously*, which is effectively *the maximal number of disjoint read quorums that can be formed*. This metric is the read-capacity. In [14, 15, 19], there is a metric called the *load* which roughly speaking measures the minimal load on the busiest site. It has a similar flavor to the metric of read-capacity. However, [15] considers only a collection of quorums every two of which intersect.

A protocol with a high read-capacity can handle heavy workload without much deterioration in response time and system throughput. However, high read-capacity does not guarantee competitive performance under light workload conditions and does not guarantee high availability. Therefore, a good protocol should simultaneously satisfy all the criteria of high read-capacity, small best case quorum sizes, high availability, and low cost of one node failure.

In this paper, we present such a new protocol, which is called **diamond quorum consensus** or simply the **diamond protocol**, for managing replicated data. In this protocol, the sites in the network are logically organized into a two-dimensional diamond structure. This protocol can be viewed as a specialized version of the grid protocol [10] because the logical structure can be seen as a grid with holes. It has been noted in [11] that grids with holes often produce a higher availability than solid grids. We show here that there is much more to the story. The diamond protocol also resembles the *crumbling walls* protocol [15]. However, [15] does not consider read and write operations. There are two main properties of the diamond quorum consensus that make it a good choice for replicated data management:

1. Compared with the majority quorum, tree quorum and grid quorum (without *holes*) protocols, the diamond protocol results in the greatest number of disjoint read quorums which shall lead to a better throughput and response time.

2. The protocol achieves the smallest optimum read quorum size and the second smallest write quorum size among the above protocols. Since the quorum size is a good indicator of messaging cost, and read operations are usually in higher proportion than write operations, this is a very desirable property.

Other than the above, we also show that diamond quorum consensus has high availability when the probability of site failures is reasonably low, and it has a low cost of node failures.

Partition failure can be a problem in a replicated database system, and handling partition failure has been a main motivation for the introduction of quorum consensus in the earlier work. However, few recent work in quorum consensus discuss partition failure handling, which in many cases will depend on the quorum consensus protocol. We shall examine the use of the *generalized virtual partition protocol* with the diamond protocol to handle partition failure.

This paper is organized as follows. In the next section, we present the diamond quorum consensus. Analysis of our protocol and comparison with other protocols are given in Section 3. The handling of partition failures is discussed in Section 4. Section 5 is a conclusion.

2. Diamond Quorum Consensus

In our protocol, the sites in a network are logically arranged in a two-dimensional diamond shape as in Figure 1. Often we shall refer to the physical sites of the network as *sites*, and refer to the nodes in the logical structure as *nodes*. When this distinction is not relevant, we shall use the terms sites and nodes interchangeably.

In Figure 1, nodes are represented by circles. The top and bottom rows of the diamond shape contain two nodes each. In a diamond structure, suppose the number of rows is odd, we label the rows from the top row to the middle row by *levels*, so that the top row is at level 0, the second row is at level 1, etc. Similarly we label the rows from the bottom row to the middle row by levels, so that the bottom row is at level 0, the second row from the bottom is at level 1, etc. The number of nodes in the rows increases by a certain amount w_i for every level i for $i > 0$. If $w_i = d$ for all i, and each of the top and bottom rows contains 2 nodes, then we call the resulting diamond structure a **regular diamond structure** and denote it by $_d$. In our basic model, w_i is 2 for all i, therefore the resulting structure is a $_2$ structure.

For example, in Figure 1, there are 32 nodes, or 32 sites, in the network, the number of nodes for the rows are 2, 4, 6, 8, 6, 4, 2.

The table in Figure 2 shows some relevant figures for the diamond structure, k is any integer greater than 0. For example, the diamond structure in Figure 1 has a maximum level of 3, 7 rows and 32 nodes in total.

In the diamond protocol, read and write quorums are formed in the following ways:

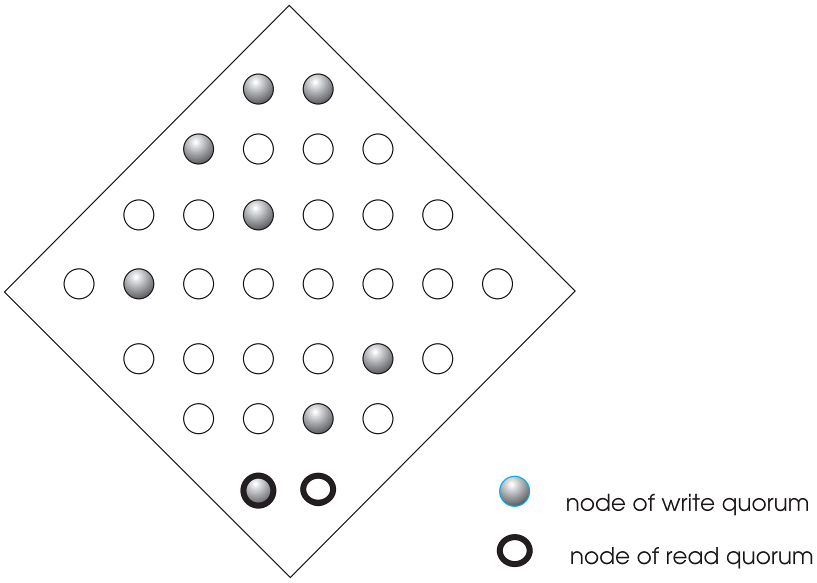


Figure 1. Example of a read quorum and a write quorum

Max Level	**Configuration**	**No. of rows**	**No. of nodes**
1	2,4,2	3	8
2	2,4,6,4,2	5	18
3	2,4,6,8,6,4,2	7	32
4	2,4,6,8,10,8,6,4,2	9	50
......	......	......	......
k	2,4,....,$2(k+1)$,....,4,2	$2k+1$	$2(k+1)^2$

Figure 2. Some figures for the diamond structure

- **Write Quorum**
  To form a write quorum, we can choose all nodes of any one row plus an arbitrary node for each remaining row.

- **Read Quorum**
  To form a read quorum, we can choose

  1. any entire row of nodes, or
  2. an arbitrary node of each row.

Figure 1 and Figure 3 show some quorums forming instances. It is obvious that the above method ensures that each write quorum intersects each other write quorum and each read quorum. We shall refer to the above protocol as **diamond quorum consensus** or simply the **diamond protocol**.

For the diamond protocol, the minimum read quorum size is obtained by choosing the whole top or bottom row of nodes. The minimal write quorum size is obtained by choosing the whole top or bottom row of nodes plus a node for each remaining row. Therefore, if k is the maximum level in a diamond structure, the minimal read quorum size is 2 and the minimal write quorum size is $2k + 2$.

In the more general case, the number of rows in a diamond structure can be even or odd, the difference in the number of nodes between 2 adjacent rows in the diamond structure can be any integer, and the number of nodes in the top and bottom row can be other than 2. In the extreme case where $w_i = 0$, for all i, and level 0 contains only one node, the diamond reduces to a single column of nodes, and we shall have the read-one-write-all protocol.

In another generalization, the diamond logical structure can be adjusted according to the number of sites in a particular network. That is, it is not restricted only to the numbers as shown in the Table 2. For instance, if there are 40 sites in a network, we can use a structure with 4 levels, 9 rows and 50 nodes, with 10 of them not being occupied by physical sites. The idea is similar to that of a *hollow grid structure* [11]. If the total number of nodes in the network does not fit into the shape above, the protocol can also accommodate the change of the shape by the addition or deletion of a number of nodes in any row and also by the addition or deletion of rows. Therefore, we have the following definition of a general diamond structure.

Definition 1. A **general diamond structure** (also called a ^{G} structure) is a stack S of n rows of nodes. Let us label the rows by numbers 1 to n. We say that a row A is above (below) another row B if the label of A is less (greater) than the label of B. For each row R in S such that no row contains more nodes than R, if we consider the rows in the stack S above and including R, each row has a size greater than or equal to the size of the row above it. If we consider the rows in the stack S below and including R, each row has a size greater than or equal to the size of the row below it.

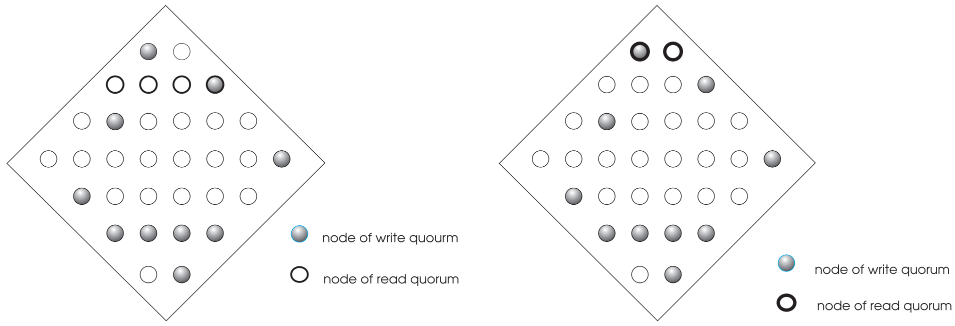


Figure 3. More examples of read and write quorums

2.1. Some Properties of the Diamond Protocol

In the $_d$ structure, for the top k rows, a row at level $i+1$ has w_i more nodes than a row at level i. If w_i is set to a constant d, then the sum of nodes for k rows from levels 0 to $k-1$ is given by a finite arithmetic series:

$$\begin{aligned} A_s(k,d) &= \sum_{i=0}^{k-1} a_i, \qquad \text{where } a_i = a_0 + id \text{ and } a_0 = 2. \\ &= k(2+kd)/2 \end{aligned}$$

Then, let $\sigma(k,d)$ be the number of nodes in a $_d$ structure with a maximum level of k, and with $w_i = d$ for all i,

$$\begin{aligned} \sigma(k,d) &= 2A_s(k,d) + (k+1)d \\ &= k^2d + 2k + kd + d \end{aligned}$$

We shall emphasize on the $_2$ structures because it will give us good performance characteristics. If k is the maximum level in $_2$, then from the above, the number of nodes is given by $\sigma(k,2) = 2k^2 + 4k + 2 = 2(k+1)^2$. Hence, if $\sigma(k,2) = N$, we have $k+1 = \frac{1}{2}\sqrt{2N}$.

The number of rows in a $_2$ structure is given by $2k+1 = 2(k+1) - 1 = \sqrt{2N} - 1$.

The number of nodes in the longest row in a $_2$ structure is given by $2(k+1) = \sqrt{2N}$.

We shall see that the above properties of a $_2$ structure lead to good performance in the next section. However, a $_2$ structure is not a general structure that can accommodate any number of sites. Therefore, we would like to discover general diamond structures that can preserve these desirable properties. This is given in the following theorem.

THEOREM 1 *Given any integer $N \geq 5$, one can build a general diamond structure (a ^{G} structure) that contains N nodes, where the number of rows in the ^{G} structure*

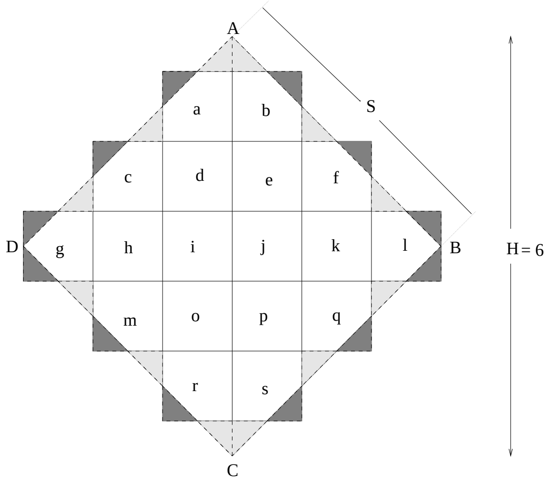


Figure 4. Hypotenuse of even length (even arrangement)

is $\left\lceil \sqrt{2N} \right\rceil - 1$*, the top and bottom rows have size 2, and the maximum row size is bounded from above by* $\left\lceil \sqrt{2N} \right\rceil$*.*

Proof: In Figure 4, the nodes in a diamond structure are represented by alphabets a to s. Each node occupies one square in the figure. Since the dark grey area is equal to the light grey area, the space occupied by all nodes is equal to the space occupied by the square $ABCD$, which has sides of length S. For the right-angled triangles ABC or ACD, the hypotenuse AC has length $H = 6$.

In Figure 5, the nodes are represented by alphabets a to x. Each node occupies one square in the figure. Let the area of each square be 1. The square $WXYZ$ has sides of length S and a hypotenuse of length $H = 7$. Since the dark grey area is equal to the light grey area, the space occupied by all nodes is less than the space occupied by square $WXYZ$ by an amount equal to the black area, which is given by 0.5.

It is easy to see that the two figures can be extended to any values of $H \geq 3$, and the relationship between the space occupied by nodes and the space occupied by the diamond $ABCD$ or $WXYZ$ will not change. The arrangement of nodes for even H is as follows: the top and bottom rows have size 2, there is only one row of maximum length, difference in size between adjacent rows is 2, Let us call this the **even arrangement**. The arrangement of nodes for an odd H is similar, except that there are two rows of maximum length, so the difference in size between the middle two rows is 0. Let us call this the **odd arrangement**.

By the Pythagoras Theorem, $H^2 = 2S^2$, or $S^2 = 0.5H^2$. Given any integer $N \geq 2$ (this corresponds to the N given sites), we can find the smallest integer H such that $N \leq S^2 = 0.5H^2$. Then, $H \geq \sqrt{2N}$, and $H = \left\lceil \sqrt{2N} \right\rceil$. If H is even (odd),

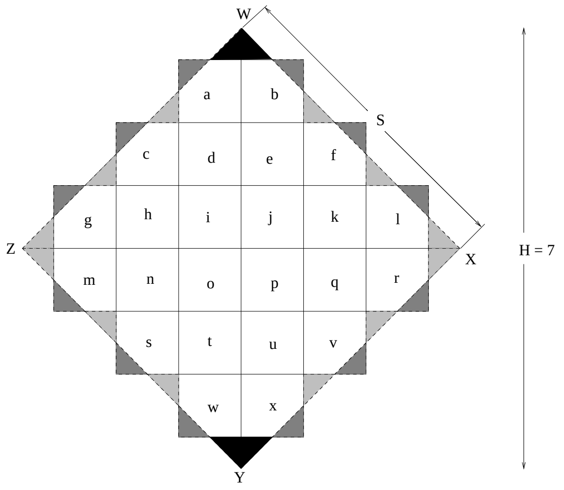


Figure 5. Hypotenuse of odd length (odd arrangement)

then we have a case as shown in Figure 4 (5), and we can match the N sites into an even (odd) arrangement. In this matching, there may be nodes that are not matched to any site.

One can see that there exists a matching such that each row contains at least 2 sites. To see this we note that N has at least $S^2 - H + 1$ nodes. This is because $0.5H^2 - 0.5(H-1)^2$ is $H - 0.5$, if $N < S^2 - (H - 0.5)$, then we should have chosen $H-1$ instead of H when looking for the *smallest integer* H as specified in the above paragraph. Therefore, there will be at most $H - 1$ *holes* in the ^{G} structure, which are the nodes not being matched to sites. For $N \geq 5$, one needs only to place at most one hole at each of the rows other than the top and bottom rows, and then place at most one more hole at the longest row.

The resulting general diamond (^{G}) structure has the same number of rows as in the full ($_2$) structure in the even (odd) arrangement, which is $H - 1 = \left\lceil \sqrt{2N} \right\rceil - 1$, and its top and bottom rows have size 2. Each row in the ^{G} structure has size less than or equal to the corresponding row in the $_2$ structure. Therefore, the longest row in the ^{G} structure contains at most $H = \left\lceil \sqrt{2N} \right\rceil$ nodes.

3. Performance Analysis

Different logical structures and quorum forming methods result in different performance in different metrics. In this section, we first introduce the metric of read-capacity. Then we examine the performance of the diamond protocol under different metrics, including the read-capacity, quorum size and availability. We compare the protocol with known protocols of majority quorum consensus, the grid protocol, the tree protocol, and hierarchical quorum consensus. For the grid protocol, we shall examine the *modified grid protocol* in [11], since it is an improvement over the original gird protocol, we shall only examine the cases without *holes* in the grid.

3.1. Read-Capacity

With the non-empty intersection property of read and write, write and write, quorums, only one write quorum can be formed at any instance in a replicated database system. Hence, the capacity analysis here is targeted only on the read operations. We define the **read-capacity** of a replicated database as:

$$\begin{aligned} \text{Read-capacity} &= \text{maximal number of concurrent read operations} \\ &= \text{maximal number of disjoint read quorums} \end{aligned}$$

Given a network with N nodes in which data is fully replicated, assume that any node of the network system can only handle one read operation at any time. In the following, we examine the read-capacity for each protocol.

- *majority quorum consensus*:
  It can only handle one read operation as there is an intersection between each pair of read quorums.

- *hierarchical quorum consensus*:
  We consider the case when the branching factor is set to 3, which is recommended in [10] as the structure that gives a minimal quorum size. In this case, each quorum is not disjoint from any other quorum, therefore it can only handle one read operation at a time.

- *tree quorum consensus*:
  For the majority tree protocol, the nodes are logically organized into a ternary tree(i.e. degree $d = 3$) of height h, i.e., each node has d children, and the maximum height is h. Each read or write quorum should have a length l and a width w, and we denote the quorum by dimensions $\langle l, w \rangle$. The protocol tries to construct a quorum by selecting the root and w children of the root, and for each selected child, w of its children, and so on, for depth l. If some node is inaccessible at depth h' from the root while constructing this tree quorum, then the node is replaced recursively by w tree quorums of height $l - h'$ starting from the children of the inaccessible node. The details can be found in [2].

  Suppose the read quorums q_r have dimensions $\langle l_r, w_r \rangle$ and the write quorums have dimensions $q_w = \langle l_w, w_w \rangle$ (see [2]). When the quorums are constructed, the following constraints should be fulfilled so that the nonempty intersection of read and write, write and write, quorums holds:

  $$\begin{aligned} l_r + l_w &> h, & w_r + w_w &> d, \\ 2 \times l_w &> h, & 2 \times w_w &> d, \end{aligned}$$

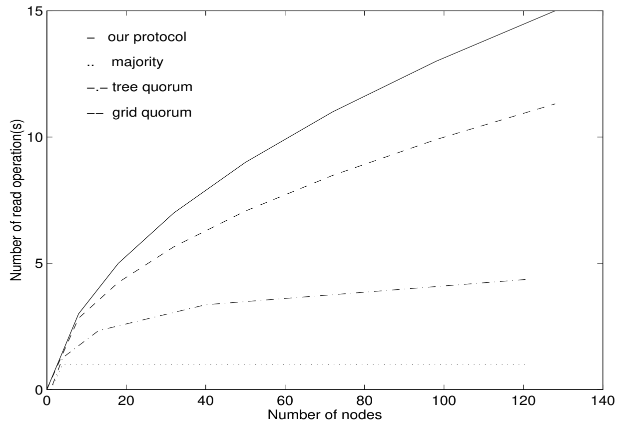


Figure 6. Comparison of read-capacities

For the comparison, we used a ternary tree model. The maximal number of disjoint read quorums occurs when the length of read quorums is 1, and each level of the tree will give rise to one disjoint read quorum. Hence it can handle at most $log_3 N$ read operations simultaneously. However, to achieve acceptable availability and overall performance, the quorum length should be set to a greater value, Hence, the maximum number of read operations that the tree quorum can handle in parallel will actually be smaller.

- *grid protocol*:
  for the grid protocol, read quorums of size approximated by $\sqrt{N}$ can be constructed without intersection. So, it can handle, on average, $\sqrt{N}$ read operations simultaneously.

- *diamond quorum consensus*:
  The maximal number of disjoint read quorums is obtained by taking a read quorum from each row of the diamond structure. From the proof of Theorem 1, there are $\left\lceil \sqrt{2N} \right\rceil - 1$ rows in a general quorum structure that has the properties stated in the theorem, hence the diamond protocol can handle $\left\lceil \sqrt{2N} \right\rceil - 1$ simultaneous read operations.

The diamond protocol can handle the maximum number of simultaneous read operation among the above protocols. That is, it can achieve the best read-capacity among the majority, tree quorum and grid protocols. The comparison is also shown in Figure 6.

Recall that the number of nodes in a diamond structure with a maximum level of k, and with $w_i = d$ for all i, is given by $\sigma(k, d) = k^2 d + 2k + kd + d$. Given a value of k, the value of $\sigma(k, d)$ increases with the value of d. Therefore, to obtain a higher read-capacity for a fixed number of nodes ($\sigma(k, d)$), one should try to obtain a greater value for k, which implies a smaller value for d [1]. However, we must also consider other factors for quorum consensus. Therefore we have chosen a value of 2 for d because with this setting, the other properties of the diamond protocol are also satisfactory.

3.2. Quorum Size

We have already discussed the importance of the quorum size. In this subsection we examine the optimal quorum sizes as well as the worst case quorum sizes. Again a number of quorum consensus are compared with the diamond quorum consensus method.

- *majority quorum consensus*:
  the read and write quorum size is approximated by $\lceil \frac{N+1}{N} \rceil$.

- *hierarchical quorum consensus*:
  As shown in [10], the minimal quorum size is given by $N^{0.63}$ in both the best case and the worst case.

- *tree quorum consensus*:
  If read quorums have dimensions $\langle l, w \rangle$ (a tree quorum of length l and width w), then write quorums must have dimensions $\langle h-l+1, d-w+1 \rangle$, where h is the height and d is the degree of the tree.

  Then, the read quorum size of tree quorum protocol varies

  $$\text{from} \quad \frac{w^l - 1}{w - 1} \quad \text{to} \quad w^{h-l} \times \frac{w^l - 1}{w - 1},$$

  where the smaller size occurs when all copies of the quorum are from the upper levels (near the root) of the tree. The size of write quorums varies

  $$\text{from} \quad \frac{(d-w+1)^{h-l+1} - 1}{d - w} \qquad \text{to} \qquad (d-w+1)^{h-l} \times \frac{(d-w+1)^{h-l+1} - 1}{d - w}.$$

- *grid quorum consensus*:
  For the grid protocol, we assume that the grid structure is approximately a square, the read quorum size is approximated by $\sqrt{N}$, and the write quorum size is approximated by $2\sqrt{N}$.

- *diamond quorum consensus*:
  The optimal read quorum size is 2 and is independent of the total number of sites. The worst case is when the longest row is chosen, or when a node from each row is chosen. From the proof of Theorem 1, the general diamond structure that satisfies the conditions in the theorem has a biggest quorum size bounded by $\left\lceil \sqrt{2N} \right\rceil$, and has $\left\lceil \sqrt{2N} \right\rceil - 1$ rows, Therefore the worst case read quorum size is $\left\lceil \sqrt{2N} \right\rceil$.

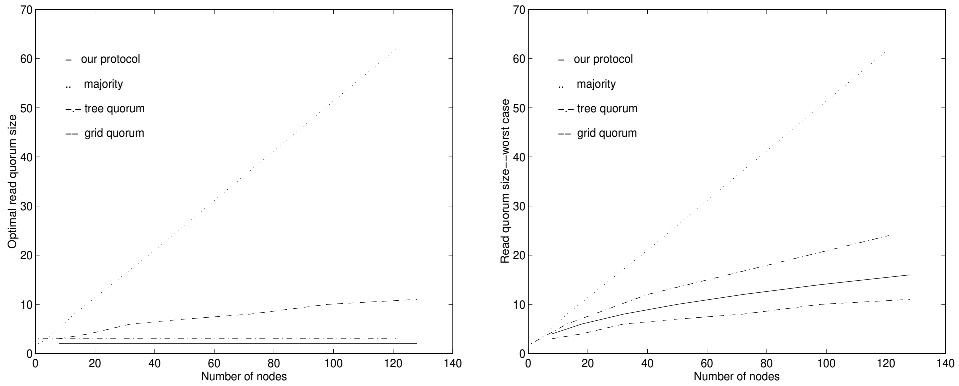


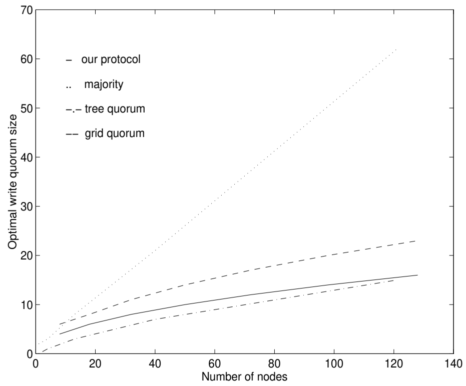
Figure 7. Read quorum size

The smallest write quorum is a node from each row union either the top or bottom row, hence the optimal write quorum size in terms of N is $\left\lceil \sqrt{2N} \right\rceil - 1 + 1.$ $= \left\lceil \sqrt{2N} \right\rceil$. In the worst case, a node from each row union the longest row forms the biggest write quorum, the size is bounded by $\left\lceil \sqrt{2N} \right\rceil - 1 + \left\lceil \sqrt{2N} \right\rceil - 1$, which is equal to $2 \left\lceil \sqrt{2N} \right\rceil - 2$,

For the diamond protocol, we can choose the top or bottom row as the read quorum, provided that they are functional, to keep the read quorum size at 2. In the other protocols under comparison, only the tree protocol can attain a comparable optimal read quorum size of 1, which is by choosing only the root node. However, in such a case, the root node becomes a bottleneck and the corresponding write quorum size will be quite big. Also, from [2], the availability will be severely affected by this choice of read quorum. [2]

Figure 7 shows the best case and worst case read quorum sizes of the *tree quorum consensus, majority quorum consensus, grid quorum consensus*, and our protocol. For the tree protocol, we set $d = 3$ because this has been chosen for performance studies in [2]. We have set the dimensions of the read quorums to be $\langle l_r, w_r \rangle = \langle 2, 2 \rangle$, and those for write quorums to be $\langle l_w, w_w \rangle = \langle h-1, 2 \rangle$. This is because from [2], $\langle 2, 2 \rangle$ is the dimensions for $\langle l_r, w_r \rangle$ that lead to the smallest read quorum size while maintaining acceptable availabilities, for the cases studied in [2].

Figure 8 shows the best case and worst case write quorum sizes of the protocols. In both cases, the diamond protocol can achieve a quorum size that is the second best and very close to best. The tree quorum has the optimal write quorum size, but its worst case write quorum size is very large. The grid quorum protocol has the smallest worst case quorum size but its optimal read quorum size and optimal write quorum size are the second highest. The comparison leads us to the conclusion that



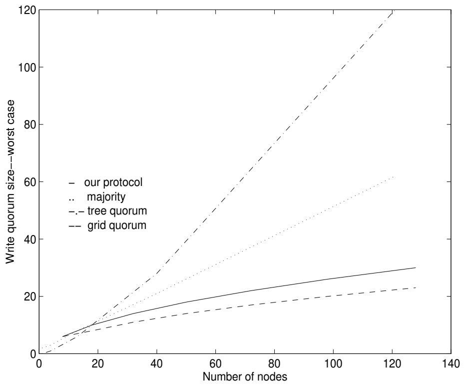


Figure 8. Write quorum size

in overall considerations of the quorum sizes, the diamond protocol is superior to the other protocols.

3.3. Availabilities

As in most previous work, the availability of an algorithm is defined as the probability of forming a quorum successfully in that algorithm. In the following we denote the probability of X by $Prob(X)$. Let p be the probability that a site is up and $q = 1 - p$ be the probability that a site is down. Let the number of sites be N.

- *majority quorum consensus*:
  Let $N = 2n + 1$, the availability of read and write operations are defined as:

$$\begin{aligned}
& Prob(majority\ copies\ are\ available) \\
& + Prob(majority + 1\ copies\ are\ available) \\
& + \ldots + Prob(majority + i\ copies\ are\ available) + \ldots \\
& + Prob(all\ copies\ are\ available)
\end{aligned}$$

$$= C_{n+1}^{2n+1} p^{n+1}(1-p)^n + \ldots + C_{n+i}^{2n+1} p^{n+i}(1-p)^{n-i+1} + \ldots + p^{2n+1}$$

- *tree quorum consensus*:
  For the tree quorum protocol, the availability can be calculated by a recurrence relations for both read and write availabilities. Let $A_h[l, w, d]$ be the availability of operations with a tree quorum of dimensions $\langle l, w \rangle$ in a tree of height h and degree d (see [2] for the definition of *dimensions*). From [2], the availability in a tree of height $h + 1$ is formulated as

$$\begin{aligned}
A_{h+1}[l, w, d] \; = \; & Prob(\text{Root is up}) \times \\
& [\text{Availability of } w \text{ subtrees with } A_h[l-1, w, d]]
\end{aligned}$$

$$
\begin{aligned}
&+Prob(\text{Root is down}) \times \\
&\quad [\text{Availability of } w \text{ subtrees with } A_h[l, w, d]].
\end{aligned}
$$

The expansion of this formula can be found in [2].

- *diamond quorum consensus*:
  The calculation of the read and write availabilities for the diamond protocol is very similar to that for the grid protocol. The diamond structure can be treated as a hollow grid, some of the positions in the grid structure are not occupied by nodes. For instance, the structure shown in Figure 1 can be regarded as an 8×7 grid structure model, with all the corner nodes being chopped off. In this particular case, 24 nodes, 6 in each corner, are being eliminated in the structure.

  The read and write availabilities of our model can be calculated in a similar way as for the *modified grid protocol* [11]. That is, a row is *alive* if at least one site in it is up, and is *dead* otherwise. A row is *good* if all sites in it are up, and is *bad* otherwise. Suppose there are n_i columns that contain m_i nodes each, for $1 \leq i \leq k$.

$$
\begin{aligned}
\text{Read availability } RA &= 1 - Prob(\text{all rows are bad}) \\
&\quad +Prob(\text{all rows are bad and alive}) \\
&= 1 - (1-p^{m_1})^{n_1} \times \ldots \times (1-p^{m_k})^{n_k} \\
&\quad +(1-p^{m_1}-q^{m_1})^{n_1} \times \ldots \times (1-p^{m_k}-q^{m_k})^{n_k}
\end{aligned}
$$

$$
\begin{aligned}
\text{Write availability } WA &= prob(\text{all rows are alive}) \\
&\quad -prob(\text{all rows are bad and alive}) \\
&= (1-q^{m_1})^{n_1} \times \ldots \times (1-q^{m_k})^{n_k} - \\
&\quad (1-p^{m_1}-q^{m_1})^{n_1} \times \ldots \times (1-p^{m_k}-q^{m_k})^{n_k}.
\end{aligned}
$$

  where there are n_k rows that contain m_k sites in the diamond structure.

- *grid protocol*
  The availabilities are also given by RA and WA, although we shall examine solid grids so that each row will have the same number of nodes.

For the above protocols, we examine 3 cases, each one with a different number of sites, for comparison of the availabilities. They are 13, 40 and 121 sites. Figures 9 to 11 show the read availabilities and the write availabilities. For Figure 9, there are 13 sites in the network. and a 3×4 grid structure is chosen for the modified grid protocol. The general diamond structure is chosen so that the row sizes are $\{2, 3, 3, 3, 2\}$. For Figure 10, there are 40 sites for Figure 10, and a 5×8 grid structure is chosen. The general diamond structure is chosen so that the row sizes are $\{2, 4, 6, 8, 8, 6, 4, 2\}$.

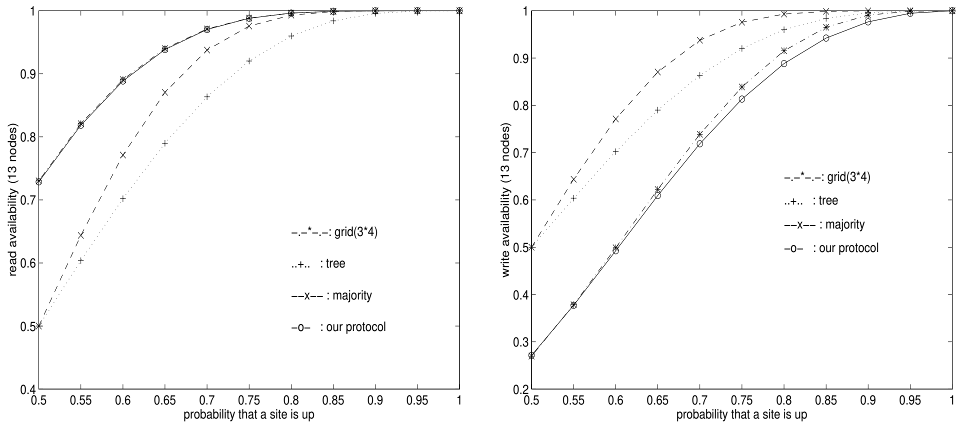


Figure 9. Availability (13 sites)

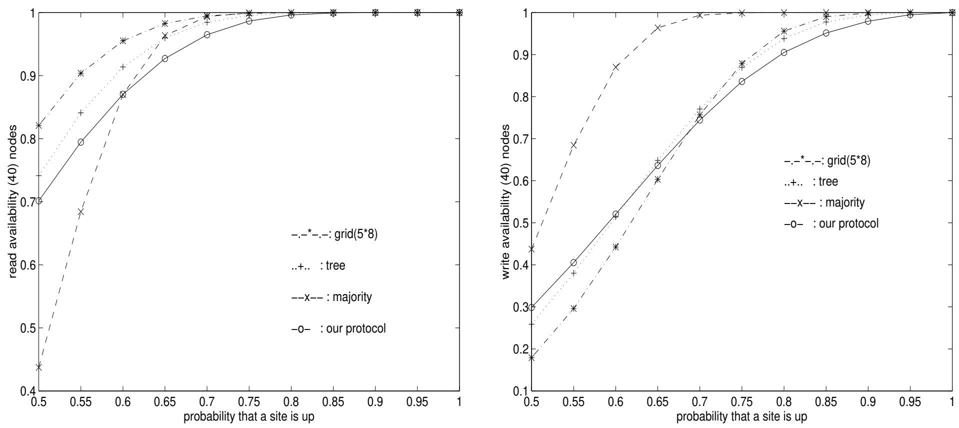


Figure 10. Availability (40 sites)

For Figure 11, there are 121 sites and a 10×12 grid structure is chosen. The general diamond structure is chosen so that the row sizes are $\{2, 4, 6, 8, 9, 10, 12, 14, 14, 12, 10, 8, 6, 4, 2\}$. Note that in all the above diamond structures, we have the properties that if N is the number of sites, then the number of rows is given by $\left\lceil \sqrt{2N} \right\rceil - 1$, the top and bottom rows have size 2, and the maximum row size is bounded from the above by $\left\lceil \sqrt{2N} \right\rceil$, as specified in Theorem 1.

From the figures, we see that when the site reliability is reasonably high, the diamond protocol performs well both in the read and write availabilities. If one would like to achieve higher write availability for the diamond quorum consensus, it is possible to do so. For example, for $N = 40$, the general diamond structure

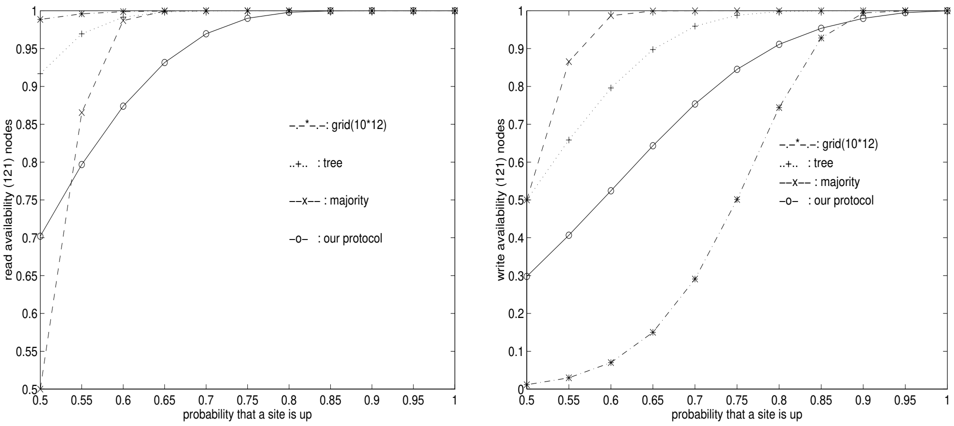


Figure 11. Availability (121 sites)

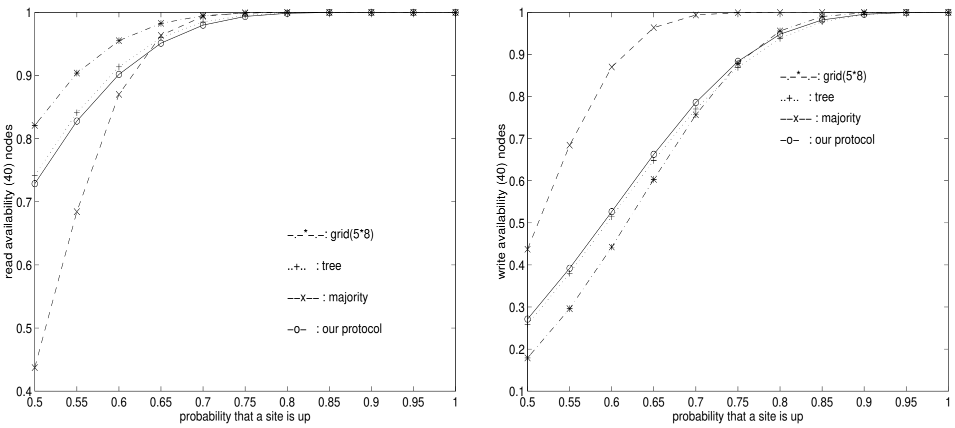


Figure 12. Improved availability (40 sites)

can be chosen so that the row sizes are { 3, 3, 6, 8, 8, 6, 3, 3}. Figure 12 shows the resulting availabilities. For $N = 121$, the general diamond structure can be chosen so that the row sizes are { 3, 3, 6, 8, 9, 10, 12, 14, 14, 12, 10, 8, 6, 3, 3}. Figure 13 shows the results. Compared with Figures 10 and 11, we can see a big improvement in the write availability when the site availability is high. In both cases, the read-capacity remains the same after the diamond structure is modified, the optimal read quorum size is increased only by one, which remains to be smaller than that of the other protocols.

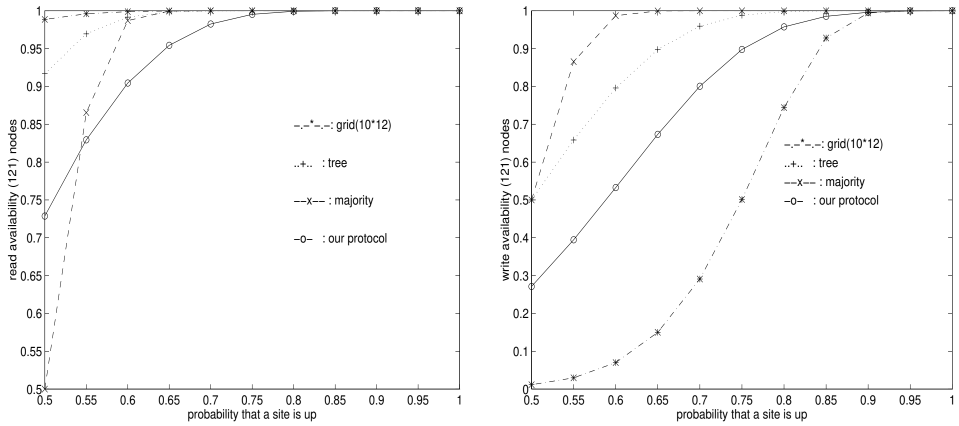


Figure 13. Improved Availability (121 sites)

4. Generalized Virtual Partition Protocol

To handle partition failures, we use *the generalized virtual partition protocol, GVP*, which is presented in [6]. This is a generalization of the virtual partition protocol (VP) in [5]. We shall give a brief description of VP and then point out the differences of GVP from VP.

We present VP in a slightly generalized form. In VP each transaction executes in a **view**. A view can be considered to correspond to the reachable part of the network as seen from a site after partition failure. Transactions executing in a view are controlled by a concurrency control protocol within the view as follows.

For each data object X, there are two positive integers, $A_r[X]$ and $A_w[X]$, called **read** and **write accessibility thresholds**, respectively, satisfying

$$A_r[X] + A_w[X] > n[X] \tag{1}$$

where $n[X]$ denotes the total number of copies of X. Thus, a set of copies of X of size $A_w[X]$ has at least one copy in common with any set of copies of X of size $A_r[X]$. Each site maintains a set of sites called its view. Views are totally ordered according to their unique **view-id**'s, which are non-negative integers.

Each copy of a data object has a **version number** $= \langle V\_id, k\rangle$, indicating that it was last written in view V with view-id V_id and that its value is the result of the kth update in that view, where $k = 0$ indicates the initial value written by the "view-update transaction" (see below). A "less than" (or "larger than") relation is defined among version numbers by their lexicographical ordering (I.e. a version number $\langle V_1\_id, k_1\rangle$, is less than $\langle V_2\_id, k_2\rangle$ if $V_1\_id < V_2\_id$ or ($V_1\_id = V_2\_id$ and $k_1 < k_2$).)

In view V, each logical data object X is assigned, if possible, read and write **quorum sizes** $q_r[X, V]$ and $q_w[X, V]$, which specify, respectively, how many copies of X must be **accessed** to, respectively, read and write X in view V. (An access operation on a copy may return only its version number, not its value.) In our terminology, a **view read(write) quorum** for data object X in view V, is a set of copies of X that can be accessed to perform logical read(write) on X in view V. $rq(X, V)$ ($wq(X, V)$) denotes the set of all view read (write) quorums for X in V. Let $n[X, V]$ be the number of copies of X that reside at sites in view V. The quorum sizes must satisfy the following conditions. For all X and V,

$$q_r[X,V] + q_w[X,V] > n[X,V] \tag{2}$$
$$1 \leq q_r[X,V] \leq n[X,V] \tag{3}$$
$$A_w[X] \leq q_w[X,V] \leq n[X,V] \tag{4}$$
$$2q_w[X,V] > n[X,V] \tag{5}$$

These ensure that each view write quorum for X in V, if any, has at least one copy in common with each view read quorum for X in V (by Equation (2)) and with any view which has at least $A_r[X]$ copies of X (by Equations (1) and (4)). If there are at least $A_r[X]$ copies of X in view V, then we say that X is **inheritable** in V. If $n[X, V] < A_w[X]$, then there is no choice for $q_w[X, V]$ which satisfies Equation (4) above; in this case both $rq(X, V)$ and $wq(X, V)$ will be empty.

Consider a transaction T executing at a site s having view V with view-id V_id. (In this case we say that T **executes** in V.) It can read or write copies at another site s' only if s' also has view V with the same view-id. (Ways to relax this restriction are discussed in [5].) If $rq(X, V) \neq \phi$ then the logical read operation $R_T[X]$ by transaction T executing in V with view-id V_id is implemented as follows:

1. Access all copies in a view read quorum in $rq(X, V)$ at sites having view V with view-id V_id,

2. Determine $vnmax = \langle Vidmax, k\rangle$, the maximum version number among the accessed copies,

3. If $V\_id \neq Vidmax$, then abort T, else read a copy in $rq(X, V)$ with version number $vnmax$.

Note that in [5], X cannot be read in V unless X is also inheritable in V. We relax this requirement by allowing a read operation on X in V once X has been "initialized" in V. This will make it possible for two concurrent partitions (under different views) to perform both read and write operations on the same logical data, provided that some transaction has performed a write without read on the data.

If $wq(X, V) \neq \phi$ for view V with view-id V_id, then the logical write operation $W_T[X]$ by transaction T executing in V is implemented as follows:

1. Access all copies in a view write quorum in $wq(X, V)$ at sites having view V with view-id V_id,

2. Determine $vnmax = \langle Vidmax, k\rangle$, the maximum version number among the accessed copies, and

3. Update the copies in a view write quorum in $wq(X, V)$ and change their version numbers to $\langle V\_id, k+1\rangle$, if $V\_id = Vidmax$ and to $\langle V\_id, 1\rangle$ if $V\_id > Vidmax$.

A site may change its view from time to time. For example, a site may want to change its view when it notices a difference between its current view and the sites it can actually communicate with. Whenever a site s changes its view to a new view, s must execute a **view-update** transaction that updates data object copies stored at site s. Site s may decide on the members of a new view V based on its own information, in which case s is called the **initiator** of V. It may also decide to use a view V initiated by another site, in which case, s **adopts** view V.

Sites change their views automatically as follows. (For details, see [5]). An initiator s of a new view V first assigns to V a unique view-id, new_view_id, that is larger than any other view-id that s has encountered. (Uniqueness of the view-id can be achieved by using the initiator's unique site ID (identification number) to be the least significant digits of the view-id.) Site s then executes a view-update transaction to update the local copy of each data object inheritable in V. For each such data object X, the view-update transaction reads the copy of X at $S'(X)$, where $S'(X)$ denotes a site in V that has a copy of X with the largest version number among a set of $A_r[X]$ copies. The version number of X is set to be $\langle new\_view\_id, 0\rangle$. If the view-id of $S'(X)$ is not larger than new_view_id, then the value of X is copied from $S'(X)$; otherwise, the view-update transaction is aborted. When this is repeated for all inheritable data objects X, the new view is **installed** at s. If a site s' accessed by the view-update transaction has a view-id less than new_view_id, then s' immediately adopts new_view_id. If a site accessed by the view-update transaction has a view-id greater than new_view_id, then the view-update transaction is aborted, in which case site s immediately adopts the greater view-id and initiates a view-update transaction with that view-id.

Some comments are in order for the case where X is not inheritable in V, since unlike [5], we may still allow reading of X in V. After V has been installed at some sites, but before any user transaction is executed in V, the copies of X at these sites, if any, have version numbers $\langle V'\_id, k\rangle$ such that $V'\_id < V\_id$. At this time, no user transaction should be allowed to read X. However, X can be written if $wq(X, V) \neq \phi$, and the first write on X in V will initialize X in V, by changing the version number of the updated copies to $\langle V\_id, l\rangle$. Thereafter, X can be read in V.

We now describe the Generalized Partition Protocol, GVP. The major difference of GVP from VP is that the definition of a quorum is in terms of a set of copies rather than by the size of the quorum. This gives us greater flexibility in defining the quorums. We shall not give the full description of the protocol here, instead, we shall list the main features that are dependent on the quorum consensus method, and explain how to incorporate the protocol in the case of diamond quorum consensus.

As in VP, in GVP each transaction executes in a *view* which is a subset of the set of all replication sites. Each view has a unique view-id, *V_id*. Each copy of a data object has a **version number** $= \langle V\_id, k\rangle$, indicating that it was last written in view V with view-id *V_id*, and that its value is the result of the $k-th$ update in that view. Transactions executing in a view are controlled by a concurrency control protocol within the view. The following are some major features.

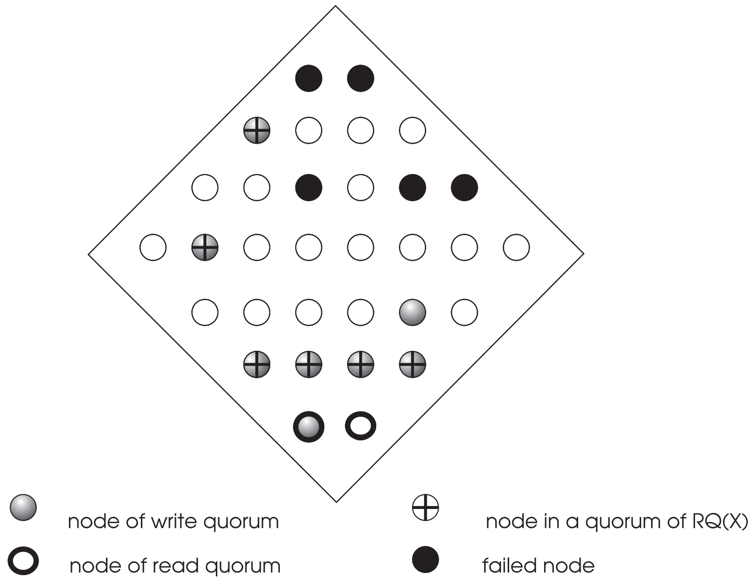


Figure 14. An example of global read, view read, and view write quorums

1. **Global read quorums** : For each data object X, a global read quorum set $RQ(X)$ is defined. X is **inheritable** in view V if V contains a global read quorum belonging to $RQ(X)$.

2. **View quorums** : For each data object X, and each view V, a **view read quorum** set $rq(X,V)$ and a **view write quorum** set $wq(X,V)$ are defined. Each view write quorum in $wq(X,V)$, if any, intersects each quorum in $RQ(X)$ and each view read quorum in $rq(X,V)$, if any. If $wq(X,V) \neq \phi$ $(rq(X,V) \neq \phi)$, then X is said to be **writable (readable)** in V.

3. **Reading and writing** : A logical write of X by a user transaction T executing in view V with view-id V_id is required to update or initialize all copies in a view write quorum in $wq(X,V)$ (where each copy has a version number that contains a view-id V_id), giving them the same new version number $\langle V\_id, k\rangle$ larger than their previous version number. A logical read of X by user transaction T is allowed in view V only if at least one copy of X accessed by the read has been *initialized* in V. A logical read of X by T reads a view read quorum from sites that have the same view as T and takes the value of the copy with the highest version number.

For our protocol, GVP is adopted as follows: if a view V is $\{s_1, s_2, ..., s_n,\}$, $RQ(X)$ can be chosen as follows:

Choose a set $\{R_1, R_2, ..., R_k\}$ which is a subset of the set of rows in the diamond structure. We call R_i a row in $RQ(X)$. $RQ(X)$ is the set of all sets of the following form:

$$
\begin{aligned}
&\{R_1 \cup \{e_{2_1}, e_{3_1}, ..., e_{k_1}\}\} \;\; where \;\; e_{i_1} \in R_i \\
&or \;\; \{R_2 \cup \{e_{1_2}, e_{3_2}, ..., e_{k_2}\}\} \;\; where \;\; e_{i_2} \in R_i \\
&\vdots \\
&or \; \{R_k \cup \{e_{1_k}, e_{2_k}, ..., e_{k-1_k}\}\} \;\; where \;\; e_{i_k} \in R_i
\end{aligned}
$$

and if V contains a quorum Q in $RQ(X)$, then

1. $rq(X, V)$ is the set of any whole row of sites in V, and
2. $wq(X, V)$ is the set of
   (i) all nodes of any one row, in V,plus
   (ii) an arbitrary node for each complete row, in V, and each row in Q.

- As $rq(X, V)$ is in V, $rq(X, V)$ must intersect any view write quorum in view V, and
- as $wq(X, V)$ contains a whole row in V, it must intersect any view write quorum in V. In addition, as $wq(X, V)$ contains one node in each row in Q, $wq(X, V)$ must intersect any quorum in $RQ(X)$.
- All quorums in $RQ(X)$ intersect each other.

An example of a possible set of global read quorum, view read quorum and view write quorum is shown in Figure 14. The black nodes are the unreachable sites for the non-black nodes, so they may be considered as a partition. The set of non-black nodes will form a view for the functional sites. The example shows the intersection properties among the quorums.

For a diamond structure with a maximum level of k, we may choose the *second*, k-th and $(2k-2)$-th row to be the rows in $RQ(X)$ because the *second* and $(2k-2)$th can minimize the cost of $rq(X, V)$, the k-th row is the row that contains the largest number of node within one row. Choosing this row can provide a high tolerance ability of partition failure.

5. Conclusion

We summarize our major findings in the table in Figure 15. In the table, N is the number of sites. The quorum size (best case) refers to the smallest read or write quorum size. The quorum size (worst case) refers to the greatest read or write quorum size. Note that although the minimal quorum size of the tree protocol is 1, the resulting protocol will have problems in read-capacity, write quorum size and also availability.

The cost of one node failure refers to the effect of a single node failure on the consensus of the smallest quorum. Suppose we have decided on a smallest quorum, but one of the nodes in the quorum has failed, then we need to form another quorum, and this cost refers to the number of additional number of nodes one needs to access. Therefore, for the diamond protocol, the cost of one node failure is the number of nodes in the second smallest quorum, which is 2.

Protocol	read-capacity	quorum size (best case)	quorum size (worst case)	cost of one node failure (worst case)
Majority	1	$\lceil \frac{N+1}{2} \rceil$	$\lceil \frac{N+1}{2} \rceil$	1
HQC	1	$N^{0.63}$	$N^{0.63}$	1
$\sqrt{N}$ Alg.	1	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$
Tree	$log_3 N$	1	N	1
Grid	$\sim \sqrt{N}$	$\sim \sqrt{N}$	$\sim 2\sqrt{N}$	$\sim \sqrt{N}$
Diamond	$\left\lceil \sqrt{2N} \right\rceil - 1$	2	$2\left\lceil \sqrt{2N} \right\rceil - 2$	2

Figure 15. Properties of some quorum consensus protocols

From the table and our previous discussions, we have shown that the diamond quorum consensus is a good choice for high throughput and efficiency in replicated data management.

There are two main contributions in this paper. First we define a new metric of *read-capacity* which we show to capture the characteristics of workload-capacity, which measures how well the system can maintain high throughput and low response time under heavy workload. Secondly we propose a new protocol, the diamond protocol that can achieve high read-capacity, low quorum size, and other desirable features for replicated data management. We also show that the new protocol can easily merge with the generalized virtual partition failure protocol.

One open problem that should be investigated is the quorum selection strategy. We suggest to use a random selection since it requires no overhead. We believe that it should be able to distribute the workload quite well during a busy period. By such a selection method, and with the proposed protocol, the chance of a node having no workload during a busy period would be small. Hence the throughput should be high and the average response time can be low. Another possible strategy for busy periods is for each site to pick the quorums in a round robin fashion. For example, if there are k quorums $Q_1, ..., Q_k$, then if the previous quorum chosen at site s is Q_i, the next quorum to be used will be $Q_{(i \mod k)+1}$. In this way, we ensure that all quorums have similar chances of being chosen and hence the workload will also be distributed quite evenly. We can try to confirm these expectations by more detailed analysis and experiments.

Acknowledgements

The authors would like to thank the anonymous referees for their very thorough review and very helpful comments which enhance the paper significantly. This research was supported by the RGC (the Hong Kong Research Grants Council) grant UGC REF.CUHK 495/95E.

Notes

1. Note that when the top row size is 1 and d is 0, we have the read-one-write-all protocol, which has the maximal read-capacity of N. However, this protocol has problems such as poor write availability.
2. This is a main reason why we have not set the upper and lower rows of the diamond to have size one.

References

1. D. Agrawal and A. El-Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 9(1):1–20, Feb 1991.
2. D. Agrawal and A. El-Abbadi. The generalized tree quorum protocol: An efficient approach for managing replicated data. *ACM Transactions on Database Systems*, 17(4):689–717, Dec 1992.
3. H. Chang and S. Yuan. Performance characterization of the tree quorum algorithm. *IEEE Transaction Parallel and Distributed Systems*, 6(6):658–662, June 1995.
4. S.Y. Cheung, M.H. Ammar, and M. Ahamad. The grid protocol: a high performance scheme for maintaining replicated data. In *Proc. of the IEEE 6th International Conference on Data Engineering*, pages 438–445, 1990.
5. A. El-Abbadi and S. Toueg. Maintaining availability in partitioned replicated databases. *ACM Transactions on Database Systems*, 14(2):264–290, June 1989.
6. A. Fu. *Enhancing Concurrency and Availability for Database Systems.* Ph.D. Thesis, Simon Fraser University, April 1990.
7. A. Fu. Delay optimal quorum consensus for distributed systems. *IEEE Transaction Parallel and Distributed Systems*, 9(3):59–69, January 1997.
8. A. Fu, T. Lau, G. Ng, and M.H. Wong. Hypercube quorum consensus for mutual exclusion and replicated data management. *Computers and Mathematics with Applications, An International Journal*, 36(5):45–59, January 1998.
9. R.A. Golding and D.E. Long. Quorum-oriented multicast protocols for data replication. In *8th IEEE International Conference on Data Engineering*, pages 490–497, 1992.
10. A. Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Transactions on Computers*, 40(9):996–1004, 1991.
11. A. Kumar, M. Rabinovich, and R. K. Sinha. A performance study of general grid structures for replicated data. In *13th International Conference on Distributed Computing Systems*, pages 178–185, 1993.
12. Y.C. Kuo and S.T. Huang. A geometric approach for consructing coteries and k-coteries. *IEEE Transaction Parallel and Distributed Systems*, 8(4):402–411, April 1997.
13. M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, 1985.
14. M. Naor and A. Wool. The load, capacity and availability of quorum systems. In *Proc. of the IEEE 35th Symposium on Foundations of Computer Science*, pages 214–225, 1994.
15. D. Peleg and A. Wool. Crumbling walls: A class of practical and efficient quorum systems. In *Proc. of the ACM 14th Symposium on Principles of Distributed Systems*, pages 120–129, August 1995.
16. S. Rangarajan, S. Setia, and S.K.Tripathi. A fault-tolerant algorithm for replicated data management. *IEEE Transaction Parallel and Distributed Systems*, 6(12):1271–1282, Dec 1995.

17. D. Saha, S. Rangarajan, and S.K. Tripathi. An analysis of the average message overhead in replica control protocols. *IEEE Transaction Parallel and Distributed Systems*, 7(10):1026–1034, October 1996.
18. R.H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, 1979.
19. A. Wool. Quorum systems for distributed control protocols. In *PhD thesis, Dept. of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Israel*, 1996.
20. S. Yuan and H. Chang. Message complexity of the tree quorum algorithm. *IEEE Transaction Parallel and Distributed Systems*, 6(8):887–890, Aug 1995.