A Novel Quorum Protocol for Improved Performance

A. Parul Pandey¹, B. M Tripathi²

¹Computer Science, Institution of Engineering and Technology, Lucknow, U.P., India

Abstract—In this paper, we present an efficient quorum protocol for reading data with minimum read quorum size. This protocol for managing replicated data is named as Wheel Quorum Protocol. We impose a logical wheel structure on the set of copies of an object. The protocol ensures minimum read quorum size of one, by reading one copy of an object while maintaining acceptable size of write operations. In this paper, we also analyze several quorum types in terms of quorum size and message overhead. Our protocol proves to incur minimum communication overhead. Wheel structure has a wider application area as it can be imposed in a network with any number of nodes. This protocol is especially beneficial for read intensive applications.

Keywords: Replica-control, distributed database, quorum consensus.

1. Introduction

In a distributed database system, data is replicated [1] to achieve fault-tolerance. One of the most important advantages of replication is that it masks and tolerates failures in the network gracefully and increases availability. In particular, the system remains operational and available to the users despite failures. In case of multiple access a problem that must be solved while using replication is about maintaining the copies in a consistent state. To keep logical data consistent, there must exist a control protocol responsible for synchronizing the access. A popular method for maintaining consistency of replicated data is weighted voting [2] which is a generalization of the majority consensus method presented in [3]. In the quorum consensus (QC) [4] algorithm, we assign a non-negative weight to each copy \( x_A \) of \( x \). We then define a read threshold \( RT \) and write threshold \( WT \) for \( x \), such that both \( 2WT \) and \( (RT + WT) \) are greater than the total weight of all copies of \( x \). A read (or write) quorum of \( x \) is any set of copies of \( x \) with a weight of at least \( RT \) (or \( WT \)). 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 [5], diamond [6], ring [7], triangular mesh [8], and grid [9] structures. A geometric approach for dealing with logical structures is proposed in [10].

In this paper we propose a novel protocol, which is called The Wheel Quorum Consensus Protocol or simply The Wheel Protocol, for managing replicated data. In this protocol, the sites in the network are logically organized into a wheel structure. This protocol can be viewed as specialized version of ring and tree protocol. As compared to tree, grid, diamond and mesh protocol, wheel protocol is very flexible in arranging nodes in a network into the logical structure. Any number of nodes can be easily organized into a wheel structure.

The paper is organized as follows. In Section 2 we describe the system model. Section 3 discusses wheel quorum protocols which elaborates the motivation behind it, wheel structure and its quorum construction for read and write. In Section 4, we present performance evaluation and section 5 discusses the related work. We conclude the paper in Section 6.

2. Model

A distributed system consists of a set of distinct sites that communicate with each other by sending messages over a communication network. No assumptions are made regarding the speed, connectivity, or reliability of the network. It is assumed that sites are fail-stop [11] and communication links may fail to deliver messages.

Replication of data is achieved by storing copies of the same logical data item at different nodes. Read and write operations can be performed on replicated data. A node needs to obtain permission from a number of copies (quorum) before performing the operation using a control protocol.

In a replicated database, copies of an object may be stored at several sites in the network. Multiple copies of an object must appear as a single logical object to the transaction. This is termed as one-copy equivalence [12] and is enforced by the replica control protocol. The correctness criteria for replicated databases is one-copy serializability [12], which ensures one-copy equivalence and serializable execution of transactions. In order to ensure one-copy equivalence, a replicated object \( z \) may be read by reading a read quorum of copies, and it may
be written by writing a write quorum of copies. The following restriction is placed on the choice of quorum assignments:

**Quorum Intersection Property:** For any two operations $o[Z]$ and $o[x]$ on a data item $x$, where at least one of them is a write, the quorums must have a nonempty intersection.

Version numbers or timestamps are used to identify the current copy in a quorum. Each node is logically characterized by few attributes as shown in figure 1. **ID** which is a unique sequential ID. In our discussion, IDs are numbered as 0, 1, 2, 3, ..., $n$. **Node_Location** is the location where the node is physically residing. In other words this is the address of a node in the network. **HUB** contains the ID of the node in the wheel which is currently acting as hub. In our discussion, ID of the HUB node is 0. **Suc** contains the ID of the successor $w_{i+1}$, which is the next node in the wheel. **Pred** contains the ID of the predecessor $w_{i-1}$, which is the previous node in the wheel.

<table>
<thead>
<tr>
<th>ID</th>
<th>Node_Location</th>
<th>HUB</th>
<th>Suc</th>
<th>Pred</th>
</tr>
</thead>
</table>

Fig. 1: Wheel Structure

The election quorum ensures that the HUB’s ID is always 0.

### 3. Wheel Quorum Protocol

#### 3.1 Motivation

Tradeoff between the cost for reading, writing, data availability and node fault tolerance is the deciding feature of all existing control protocols for replicated data. For example, the read-one write-all scheme needs only one copy as read quorum, but has the convenience of having a write quorum equal to the total number of copies (thus not tolerating a single node of failure).

The main motivation for our work was to develop a protocol which had a constant minimum cost for reading, while maintaining an acceptable cost for writing, since we are interested in systems where read operations are much more frequent than write operations.

To achieve this property, a logical wheel structure will be imposed on the set of copies of the object. This structure is used by operations to determine the copies that must be read or written. Figure 2, represents 5 nodes arranged in a wheel structure. Wheel logical structure can be arranged on any number of nodes, whereas other logical structures have constraints with nodes arrangement. We note that this structure is logical, and does not have to correspond to the actual physical structure of the network connecting the sites, storing the copies. This wheel structure is used to motivate the protocol.

#### 3.2 The Wheel Structure

Let $W_n = w_0, w_1, w_2, \ldots, w_{n-1}$ be the set of nodes that store copies of a replicated data item. A wheel, $W_n$, is a logical structure with $n$ nodes, formed by connecting a single node called HUB to all vertices of an $(n-1)$ cycle. The numerical notation for wheels is used inconsistently in the literature: some authors instead use $n$ to refer to the length of the cycle, so their $W_n$ is the graph we would denote as $W_n+1$. All nodes in the cycle maintain adjacency relationship by maintaining ID’s of their successor and predecessor. Each node is defined by attributes ID, Node_Location, HUB, Suc, and Pred as shown in figure 1. Wheel structure is easily imposed on the set of nodes by selecting first node as HUB and adding other nodes as spokes in cycle by defining the successor (Suc($i$)) and predecessor (Pred($i$)) operations and by setting HUB in each spoke. Other operations are GetPermission($i$) and rand(1..n).

GetPermission($i$), returns TRUE if the node $w_i$ allows access to its own copy of the item. GetPermission($i$) returns FALSE when either node $w_i$ refuses access or cannot be contacted due to failure. rand(1..n) selects and returns random number from 1 to $n$, where $n$ is the number of nodes in wheel. This random number represents ID of selected node.

![Fig. 2: Wheel Structure](image)

#### 3.3 The Wheel Protocol

In this protocol, all copies of a replicated data item are organized into a wheel structure. Specific algorithms are used for read and write quorums construction. There is one election algorithm for electing new HUB in case
of failure of HUB or in case load threshold exceeds its limit. These algorithms use the adjacency information to guarantee quorum intersection, and to maintain the quorum sizes small. There are three type of quorums, Read, Write, and Election quorum.

**Read Quorum** is formed by getting access permission from HUB.

**Write Quorum** is obtained by getting access permission from HUB and half of alternating nodes in the cycle, thus requiring the majority of the total number of copies. As an example, consider a replicated data item with six copies arranged in a wheel structure as shown in figure 3. Eligible read quorum is 0 (i.e. HUB) and sets eligible for write quorum are: \{0,1,3,5\}, \{0,1,2,4\}, \{0,3,5,2\}, \{0,4,1,3\} and \{0,5,2,4\}.

Notice that eligible quorums are coteries, satisfying the minimality and intersection properties\(^1\).

**Fig. 3**: 6 copies organized into a logical structure

Election quorum is called in two situations

1) When HUB crosses its load threshold
2) When HUB is unavailable

In both the above cases, the node initiating election quorum algorithm, selects randomly any 2 adjacent nodes, checks their version and makes the latest one the HUB by changing the location address between the old HUB and the newly elected one. This logically swaps the location of the two nodes. Other nodes are unaffected as they identify HUB by its ID, which is 0. Only the node location is changed.

Advantages of this Election Quorum are-

1) HUB is never overloaded, as it gets swapped with a latest node whenever load threshold crosses its limit.
2) Improved load distribution. Assuming that each node in cycle has the equal probability of being selected as a new HUB, no node will be working as HUB for a longer time.

3) Constant minimum possible Read Quorum size of one. As, even if HUB is failed, it will be replaced with a new HUB. Thus ensuring that a request always reads data from HUB.

Without using election quorum, in the failure of HUB, Read Quorum can be achieved by accessing any 2 adjacent nodes in the cycle, which is double the cost of doing it with HUB. Our system has more number of reads as compared to write, so reads will keep on costing double till HUB recovers. All this can be avoided by using election quorum and electing new HUB. This way, present as well as subsequent reads can be satisfied by reading only HUB.

### 3.3.1 Quorum Construction

There are three algorithms for the wheel protocol. Algorithm 1, 2, 3 for read, write and election quorum respectively.

Algorithm 1 defines read quorum construction. This algorithm returns the HUB as the read quorum. In case of HUB failure, the new HUB is elected by invoking the ElectionQuorum Protocol, which uses a random node in cycle.

**Algorithm 1 Read Quorum(i)**

```plaintext
if Empty(Wheel) then
  Return(nil)
else if GetPermission(HUB) is False then
  r= rand(1 .. n)
  Get ElectionQuorum(r)
  Return(HUB)
else
  Return(HUB)
end if
```

Algorithm 2 is to find write quorum. This protocol collects majority of nodes forming quorum between nodes in cycle of wheel in list, Quorum_list[]. This Quorum_list[] along with HUB makes write quorum. Protocol tries to form write quorum with current_node by traversing the cycle until, either a quorum is obtained or all copies have been examined (in which case quorum was not obtained and the request for writing is refused). In case of HUB failure Election Quorum elects a new HUB.

In case of HUB failure, Election Quorum (Algorithm 3) elects a new HUB. Election quorum selects two adjacent nodes (using successor function), selects the node with latest value and makes it the HUB.

\(^1\)The fact that the quorums are distinct and have the same size shows that they satisfy the minimality property: the intersection property will be shown later, when providing the protocol correctness
Algorithm 2 Write Quorum(i)

- Main routine
1: nodes_covered=0
2: current_node = i
3: if GetPermission(HUB) is False then
4: \( n = \text{random(cycle nodes)} \)
5: Get ElectionQuorum(n)
6: GetPermission(HUB)
7: end if
8: if current_node is HUB then
9: current_node = rand(1..n)
10: end if
11: while Empty QuorumList[] and \( \text{nodes_covered} < n \) do
12: Quorum_list[]= Check(current_node)
13: current_node=Suc(current_node)
14: nodes_covered++
15: end while
16: Return(HUB \( \cup \) QuorumList[])

- Check(i)
1: Quorum_list[]= null
2: Fail= nodes_checked=0
3: while \( \text{Fail} \neq 1 \) and nodes_checked < \( \lfloor n/2 \rfloor \) do
4: if GetPermission(i) then
5: Quorum_list.add(i)
6: i=Suc(Suc(i))
7: nodes_checked++
8: else
9: Fail=1
10: end if
11: end while
12: if Fail then
13: Quorum_list.flushall()
14: return(Quorum_list[])
15: else
16: return(Quorum_list[])
17: end if

Algorithm 3 Election Quorum(i)

1: current_node=i
2: Quorum=0
3: nodes_done=0
4: if current_node is HUB then
5: current_node=rand(1..n-1)
6: end if
7: while Quorum is Empty or nodes_done < n do
8: if current_node is accessible then
9: if Suc(current_node) is accessible then
10: Latest_node=Node_Location with most recent value
11: Swap Node_Location of HUB and Latest_node
12: Quorum=Latest_node
13: else
14: current_node=Suc(Suc(current_node))
15: nodes_done=nodes_done + 2
16: end if
17: else
18: current_node=Suc(current_node)
19: nodes_done=nodes_done+1
20: end if
21: end while

4. Performance Evaluation

Different logical structures and quorum forming methods result in different performances in different metrics. In this section, we present performance analysis of wheel protocol under different metrics. We compare the protocol with known protocols of majority quorum consensus[3], the grid protocol[9], the tree protocol[13], the hierarchical quorum consensus[14], the diamond protocol[6], the triangular mesh protocol[8], and the ring protocol[7].

4.1 Quorum Size

In this section we examine optimal and worst case read and write quorum sizes. We analyze quorum sizes based on the type of applications where a number of read operations are much higher than the number of write. Read and write quorum sizes for majority quorum is \( \lceil \frac{N+1}{N} \rceil \). We have not included majority in our quorum comparison, but its very clear that it has larger read and write quorum sizes. Hierarchical quorum consensus has best, as well as worst quorum size of \( N^{0.63} \). In the case of the tree quorum protocol ,the size of read quorums vary from 1 to \( (d+1)^h \). On the other hand, the cost of write operations is \( [(d+1)^{h+1} - 1]/d \) \( [(d+1)^h - 1]/d \). For a grid protocol, we assume that the grid structure is approximately a square, the read quorum size is approximated by \( \sqrt{2N} \), and the write quorum size is approximated by \( 2\sqrt{2N} \). For diamond quorum consensus optimal read quorum size is 2 and is independent of the total number of sites. Worst case read quorum size is \( \lceil \sqrt{2N} \rceil \). Optimal and worst quorum sizes for diamond write quorum are \( \lceil \sqrt{2N} \rceil \) and \( 2 \lceil \sqrt{2N} \rceil - 2 \) respectively. For the special case taken in [7], read and write quorums are given by \( q_r = n^{log_d2} \) and \( q_w = ((\frac{d}{2}) + 1)log_dn \) for the ring protocol. The quorum size in three triangular-mesh
based protocols is \( k \), which is \( \lceil \sqrt{2N} \rceil \).

The wheel protocol has exceptionally minimum read quorum size of one (independent of number of sites), as read can be satisfied by reading only HUB. The remarkable point is that even in worst case the read quorum of wheel protocol remains one, by ensuring availability of HUB. Election quorum is used to elect new HUB whenever current HUB fails. Systems with more number of read operations than write get benefited by this smallest possible read quorum size and thus reduces the cost of operation. Write quorum size is \( \lceil \frac{N-1}{2} \rceil + 1 \). This large write quorum is justifiable against the constant small quorum size of one. This protocol is especially beneficial for systems with much more number of reads than write for eg. web based shopping sites.

Figure 4. shows that wheel protocol has best read quorum size in optimal as well as worst case. Diamond has comparable read quorum size in optimal case but size increases up to \( \lceil \sqrt{2N} \rceil \) in worst case. Similarly ring is comparable in worst case but not in optimal case. Figure 5 shows optimal and worst write quorum comparison.

### 4.2 Message Overhead

This section presents message overhead analysis of different quorum protocols. Analysis is based on model and message overhead relations used in [15]. For the purposes of our study, we consider only the best possible implementation (the one with the least number of messages). The proportion of update operations in the load is represented by \( w \). A small \( w \) indicates that there are generally few write operations in the system. For instance, the workload can have many queries (read-only transactions) and few update transactions. Each of these few update transactions, however, can have many...
write operations. With this, assuming that a transaction contains on average \( o_w \) write operations, the message overhead for wheel protocol are given below.

Message overhead for point-to-point is given as:

\[
msg = \frac{3w \left\lceil \frac{n+1}{2} \right\rceil}{o_w} \tag{1}
\]

Message overhead for multicast is given as:

\[
msg = w \left\lceil \frac{n+1}{2} \right\rceil + 2 + (1 - w) \tag{2}
\]

Comparison among different protocols message overhead for point-to-point is shown in figure 6(a). Minimum communication overhead is achieved by tree and wheel protocols. Overhead increases with increasing value of \( w \). Tree shows better performance than wheel protocol, whereas, its quorum size is larger than wheel protocol. Wheel protocol, ensures read quorum size of 1 even in worst case, whereas, it becomes as big as \((d+1)^h\) in tree. So, for read intensive applications, wheel gives both the advantages of smallest read quorum and smaller communication overhead.

In case of multicast figure 6 (b), wheel and tree incur minimum message overhead than other protocols. Overhead increases with increasing number of writes in transactions (i.e. \( w \)).

5. Related Work

In this section we compare the wheel protocol to other existing protocols for maintaining the consistency of replicated data. The simplest replica control protocol is the read-one write-all protocol, where a read operation is executed by reading any copy and a write, writes all copies of the object. In order to increase the fault-tolerance of write operations, voting protocols were proposed [16], [2], where write operations are not required to write all copies. In a failure free system, both the static and the dynamic protocols require read operations to access several copies. Dual Quorum [17] reduces size of both read and write quorums by not making them intersect and regular semantics are enforced by communication between both quorums. This increases its communication overhead, whereas Wheel has no overhead like this.

Finally, the notion of imposing logical structures on a network of sites has been proposed before to solve different problems. Maekawa [18] proposed imposing a logical grid on a set of sites to derive efficient \( O(\sqrt{N}) \) solutions for mutual exclusion. Agrawal and El Abbadi [19] proposed imposing a logical tree to solve the mutual exclusion problem using \( O(\log n) \) messages. This approach was extended to replica control protocols that use several logical structures imposed on set of copies. Kumar [13] constructs a logical tree on a set of copies, where the copies actually correspond to the leaves of the tree. This results in a protocol where read and write quorums are of size \( N^{0.63} \). Agrawal and El Abbadi [20] imposed a logical tree on the set of nodes and reduced read size to one when there is no fault. But read quorum size increases with failures to \( \left\lceil \frac{N+1}{2} \right\rceil \) and write quorum size is \([[(d+1)^h+1] - 1]/d\). Storm [21] proposes a flexible heterogeneous quorum based which is again based on on tree shaped voting structures. Triangular mesh protocol [8], in which the nodes in the system are organized into a triangular mesh which has a quorum size of \( O(\sqrt{2N}) \). Our protocol draws on many of these ideas, and extends them to develop an efficient and fault-tolerant replica control protocol.

The distinguishing feature of our approach is that we directly address the issue of low cost read operations,
and unlike other logical structure based approaches, the wheel quorum protocol, in a failure-free system or with failure does not require read operations to access more than one copy.

6. Conclusion

In this paper we have proposed a new fault tolerant protocol for replicated data control in which the read quorum is constant of size one, without incrementing the write quorum. The design of the protocol directly addresses one of the main problems of replicated data: the necessity of read operations to access several copies in order to ensure the fault-tolerance of write operations. In case of failure, the wheel protocol continues executing both read and write operations with a high probability, although cost of execution is higher. Wheel protocol provides smallest quorum size with minimum message overhead. Message overhead for wheel protocol [22] multicast never exceeds 2, in fact for lesser number of writes its smaller than 1.5. In particular, our protocol performs well in systems where read operations are requested more frequently than write ones.

References