# How to Select a Replication Protocol According to Scalability, Availability and Communication Overhead

R. Jiménez-Peris, M. Patiño-Martínez[*]
School of Computer Science
Technical University of Madrid (UPM)
Madrid, Spain
{rjimenez, mpatino}@fi.upm.es

G. Alonso
Department of Computer Science
Swiss Federal Institute of Technology (ETHZ)
Zürich, Switzerland
alonso@inf.ethz.ch

B. Kemme
School of Computer Science
McGill University
Montreal, Canada
kemme@cs.mcgill.ca

## Abstract

Data replication is playing an increasingly important role in the design of parallel information systems. In particular, the widespread use of cluster architectures in high-performance computing has created many opportunities for applying data replication techniques in new areas. For instance, as part of work related to cluster computing in bioinformatics, we have been confronted with the problem of having to chose an optimal replication strategy in terms of scalability, availability, and communication overhead. Thus, we have evaluated several representative replication protocols in order to better understand their behavior in practice. The results obtained are surprising in that they challenge many of the assumptions behind existing protocols. Our evaluation indicates that the conventional read-one/write-all approach is the best choice for a large range of applications requiring data replication. We believe this is an important result for anybody developing code for computing clusters as the read-one/write-all strategy is much simpler to implement and more flexible than quorum-based approaches. In this paper we show that, in addition, it is also the best choice using a number of other selection criteria.

**Keywords:** data replication, quorums, transactions.

## 1 Introduction

Data replication is playing an increasingly important role in the design of distributed information systems. In particular, the widespread use of cluster architectures in high-performance computing [6] has created many opportunities for replicated databases. In applications that can be easily parallelized (web servers, data mining, computational genomics) it is conceivable to use a series of replicated databases as the means to share up-to-date information among all the sites in a cluster. The database takes care of complex tasks such as indexing, recoverability, or concurrency control thereby making the application easier to develop and maintain.

[*] This work has been partially funded by the Spanish Research Council (*CICYT*), contract number *TIC98-1032-C03-01*.

Data replication, however, must be used carefully when performance is concerned. The most relevant strategy for cluster computing is eager, update everywhere replication (e.g., full consistency and updates can be generated anywhere in the system). Unfortunately, this strategy can be tremendously inefficient [9]. There are ways to get around the inefficiencies, at least for systems up to a given size [11, 14], but there are still many aspects of the problem that have not been studied in detail. More specifically, one could think of using quorum based strategies to achieve consistency across the cluster while still minimizing the overall cost in terms of performance penalties, communication overhead, and overall availability.

As part of work related to cluster computing in bioinformatics [3], we have been confronted with exactly this problem. We have a number of large-scale parallel computations that need to share data dynamically. The best solution is clearly to use a replicated database (for a number of reasons that are beyond the scope of this paper but that include the volume of data, the need to index it and the ability to perform complex queries). Our intuition told us that we should be able to improve the system by using quorums but it was not clear which one would provide the best results. Although there are several studies of the availability and load distribution of quorum systems [13, 12, 4, 15, 2] we could not find in the literature any indication of what type of quorum could be most suitable for our purposes. Thus, we evaluated several protocols in order to better understand the problem. Unlike previous work, our analysis is based on realistic environments. For instance, comparisons among quorum properties have been typically made on an asymptotic basis. However, asymptotic analysis for eager data replication is not very useful since this kind of replication does not scale beyond a few tens of sites. For the analysis to be meaningful, it is necessary to take into account constant and multiplicative factors that are usually discarded in an asymptotic analysis. Additionally, we take into account common database optimization strategies that substantially change the behavior of the protocols. These optimizations introduce asymmetries in the processing of transactions. Such asymmetries have not been taken into account by previous studies, which wrongly assume that transactions require the same computing resources at all sites where they are executed. The study of the influence of this asymmetry is one of the main contributions of this paper.

The results of our analyses are surprising in that they challenge the notion that quorums improve performance, availability, or communication overhead. The results indicate that the conventional read-one/write-all approach is clearly the best choice for a large range of applications requiring data replication. This is an important result since the read-one/write-all strategy is much simpler to implement and much more flexible than quorum based approaches. Thus, scientists and application developers using replicated databases in clusters can take advantage of a simpler design while still having the guarantee that they are getting the best possible behavior out of the replication protocol.

In what follows we analyze a number of replication protocols from the point of view of scalability (Section 4), availability (Section 5), and communication overhead (Section 6). For obvious reasons, we do not analyze all existing protocols but a representative subset. We consider realistic scenarios and study the effect of typical optimization strategies used in databases. For each protocol, we indicate the application for which they are best suited and then compare all of them from the point of view of cluster computing (Section 7).

## 2 System Model

A replicated database consists of a group of sites $n = \{N_1, N_2, ..., N_n\}$ which communicate by exchanging messages. Sites are fail-stop, and site failures can be detected. We consider a crash-recovery model where sites can recover and re-join the system after synchronizing their state with the one of the running replicas. The database is fully replicated, and thus, each site contains a copy of the database. We assume that all sites are homogeneous and each site is able to execute $t$ transactions per second (tps).

Clients interact with the database by issuing transactions. Transactions are executed atomically, i.e., a transaction either commits or aborts. Transactions are partially ordered sets of read $(r)$ and write $(w)$ operations. Transactions are executed atomically, i.e., a transaction either commits are aborts at all participating sites. If a transaction contains write operations, a 2-phase-commit protocol at the end of the transaction is executed among all sites. For replicated databases, the correctness criterion is one-copy-serializability [5]. In this criterion, each copy must appear as a single logical copy and the execution of concurrent transactions must be equivalent to a serial execution over all the physical copies.

A client submits a transaction to one of the sites in the system, and this site coordinates its actions with the rest of the system. A transaction is called *local* at the site it is submitted to, and *remote* at the other sites. We assume that all sites receive the same amount of local transactions. We consider in the study two kinds of transactions: queries, which contain only read operations, and update transactions, which contain both read and write operations.

## 3 Replication Protocols

Several protocols have been proposed for database replication. These protocols mainly differ in the number of sites contacted to perform a read or write operation. In this section we describe the four protocols we will analyze.

### 3.1 Read-one Write-all

Under the read-one write-all (ROWA) algorithm [5], a transaction just reads a copy of the data (the local one to minimize the communication cost), while write operations must be performed at all sites. Read-only transactions (queries) can be executed locally, while update transactions are executed at all sites in the system. Writes can be propagated immediately or they can be deferred until the transaction commits.

A naive version of ROWA would require all replicas to be available [5] to perform a write operation. A variation of this technique known as read-one write-all-available (ROWAA) [5] improves the availability of the database. From now on we will consider the later protocol.

### 3.2 Quorums

Quorums [18, 8] were originally proposed to cope with communication failures, ensuring that in case of partitions at most only one partition will run, thereby preventing inconsistencies. An additional claimed advantage of quorums has been that the cost of running a transaction is reduced with respect to ROWAA as writes are only performed in a quorum instead of in all the sites.

A quorum system is defined as a set of subsets of sites, or quorums, with pair-wise non-empty intersections. The non-empty intersection property is crucial in that it allows any quorum to take decisions on behalf of the whole system, and still guarantee overall consistency. In particular, read $(rq)$ and write $(wq)$ quorums must be such that read and write operations or two write operations on the same data item overlap. That is, any read quorum must overlap with any write quorum, and write quorums must overlap among them.

We are aware that many different variations of these protocols have been proposed in the literature. However, our goal can be accomplished by examining a small number of *canonical* protocols. Thus, for simplicity and reasons of space, in what follows we will consider only these protocols and ignore any existing variations of them. For the purposes of our study, existing variations have similar characteristics to the protocols we study (i.e., they only introduce small constant factors). In the cases were there are significant changes in behavior, this is briefly discussed in the corresponding section.

### Majority

In the majority quorum (also known as quorum consensus) algorithm [18] read and write quorums must fulfill the following constrains: $2 \cdot wq > n$ and $rq + wq > n$, being $n$ the number of sites. The minimum quorums satisfying these constraints are: $2 \cdot wq = n + 1$ and $rq + wq = n + 1$ and therefore, $wq = \lfloor \frac{n}{2} \rfloor + 1$ and $rq = \lceil \frac{n}{2} \rceil$. Thus, a write quorum can be formed by any majority, and read quorums by half of the system sites, if $n$ is even, or by a majority if $n$ is odd.

### Tree

In [1] a new kind of quorum protocol is proposed to reduce the size of write quorums while keeping a singleton read quorum (in a non-failure scenario) as in the ROWAA protocol. This is achieved by imposing a logical tree structure on the sites. This tree must be a complete tree of an odd degree. Write operations are performed on a quorum formed by the root, a majority of its children, a majority of the children of each of these children, and so forth. Hence, two concurrent writes have at least one element in common at each level of the tree. For instance, in a tree of degree 3 and three levels (Fig. 1), a write operation could access the set of copies $\{1, 2, 4, 6, 7, 11, 12\}$.

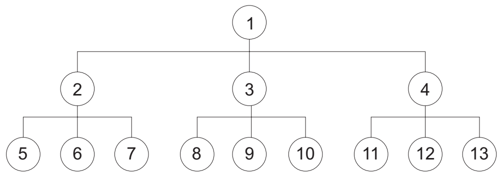


Figure 1: Tree of degree and height 3

Reads are performed on the root. If the root is not available, the read should be performed on a majority of its children. For each unavailable site needed for the majority, a read on a majority of its children should be performed, and so forth. For instance, in the tree of Fig. 1 a read could access $\{1\}$. If 1 is unavailable, it could access $\{2, 3\}$ and if 1, 3, and 4 are unavailable, it could access $\{2, 9, 10\}$. Since write operations access a majority of sites at each level, read and write operations will overlap at least in one site.

### Grid

A different kind of quorum, grid quorum, is proposed in [7]. This quorum assumes that the sites are organized in a grid of $r$ rows and $c$ columns. A read quorum consists of accessing an element of each column of the grid. A write quorum requires applying the updates in one column and locking an element from each of the remaining columns (a read quorum). Given the grid in Fig. 2, examples of read quorums are: $\{5,2,7,12\}$, $\{9,10,3,4\}$, or $\{1,6,11,4\}$. Examples of write quorums are: $\{1,5,9,2,7,4\}$, $\{2,6,10,5,7,4\}$, $\{3,7,11,9,6,4\}$.

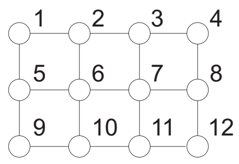


Figure 2: $3 \times 4$ Grid

Read operations overlap with writes, since read quorums contain one element from each column. That is, they contain one element from the column used for a write operation. Two write operations also overlap. Since a write quorum locks one element in each column, no other write can be concurrently performed in any column.

## 4 Scalability

### 4.1 Scalability with symmetric load

In a first approximation, we will assume that all operations have the same cost, regardless of whether they are local or remote operations. We will refer to this type of systems as *symmetric*.

Let $L$ be the transactional load arriving at the entire system. Let $L_w = w \cdot L$ be the load created by updates, with $w$ being the proportion of update operations in the load. Let $L_r = (1-w) \cdot L$ be the load created by read operations. Assume that write operations are performed in $wq$ sites (write quorum) and read operations in $rq$ sites (read quorum). Let $P_w = \frac{wq}{n}$ be the probability for a site to participate in a write quorum. Let $P_r = \frac{rq}{n}$ be the probability for a site to participate in a read quorum. With this, the load, $t$, at a site is given by the expression:

$$t = P_w \cdot L_w + P_r \cdot L_r \qquad (1)$$

which in terms of $L$ can we rewritten as:

$$t = L \cdot (w \cdot P_w + (1 - w) \cdot P_r) \qquad (2)$$

The scalability of a system is given by the total processing capacity of the entire system divided by the processing capacity of one site. In our case the total load is $L$ and the load at each site is $t$. With this, we can study the scalability in terms of the scale out factor, $so = \frac{L}{t}$:

$$so = \frac{L}{t} = \frac{1}{w \cdot P_w + (1 - w) \cdot P_r} \qquad (3)$$

### 4.2 Scalability with asymmetries in the system

Expression 3 assumes a symmetric system. However, distributed databases are rarely symmetric. The most common asymmetry is due to a load optimization strategy used to minimize redundant work. A local operation is executed by parsing the corresponding SQL statement and executing it in its entirety. The effects of the execution might be very small compared with the amount of data perused. For instance, it might be necessary to scan through a long table (i.e., read all tuples) to update just one tuple in that table. Doing this at all sites is quite inefficient. Instead, the local site can send to all other sites the key to the tuple that needs to be updated along with the new value. Remote sites only need to apply the update without having to read any data. This strategy is used, for instance, in Postgres-R [11].

In theory, this strategy can be applied to both read and write SQL statements. In practice, however, it is of limited use for read operations. Read operations are denoted as such because they do not introduce new information in the database but this does not mean that they do not introduce new data. A join, for instance, results in a temporary new relation (could also be permanent if materialized views are used). The creation of such temporary structures is governed by relational algebra, which operates on relations, not on tuples (tuples are accessed by manipulating the relation according to predicates that select the desired tuples). It is easy to rewrite an update SQL statement so that it updates only the tuples corresponding to a set of keys. It is much more difficult, and probably quite inefficient, to do the same for read SQL statements other than selection. Hence, in here we will distinguish only between local and remote writes. Read operations will be considered symmetric. With this, the probability for a site to participate in a write operation can be divided into two parts:

$$P_w = P_w^O + P_w^R$$

where $P_w^O$ is the probability of being the originator of a write transaction and $P_w^R$ is the probability of participating in a remote transaction (i.e., a transaction originated by other sites). Moreover, we assume that the cost of performing an update locally is 1 while the cost of performing a remote update is given by a variable factor $wo$ ($0 < wo \leq 1$, when $wo = 1$ the system is symmetric for write operations). With this:

$$so = \frac{L}{t} = \frac{1}{w \cdot P_w^O + w \cdot wo \cdot P_w^R + (1-w) \cdot P_r} \qquad (4)$$

## 4.3 Quorum probabilities

The probabilities of being in read or write quorums for each different protocol are summarized in the following table:

| | $P_r$ | $P_w^O$ | $P_w^R$ |
|---|---|---|---|
| ROWAA | $\frac{1}{n}$ | $\frac{1}{n}$ | $\frac{n-1}{n}$ |
| Majority | $\frac{1}{2}$ | $\frac{1}{n}$ | $\frac{1}{2}$ |
| Tree(3) | $\frac{1}{n}$ | $\frac{1}{n}$ | $\frac{2 \cdot (2^{\lfloor log_3 n \rfloor} - 1)}{n} \sim \frac{2 \cdot n^{0.63}}{n}$ |
| Tree(d) | $\frac{1}{n}$ | $\frac{1}{n}$ | $\sim \frac{d+1}{d-1} \cdot \frac{n^{log_d \frac{d+1}{2}} - 1}{n}$ |
| Grid | $\frac{1}{\sqrt{n}}$ | $\frac{1}{n}$ | $2 \cdot \frac{\sqrt{n}-1}{n}$ |

We assume that the originator of a transaction submits it to a quorum of sites to which it belongs. In the case of $P_r$, the probabilities are given by the size of the read quorum divided by the number of sites in the system. In the case of $P_w^R$, it is the size of a write quorum minus one divided by the number of sites since, since by definition a site in a remote quorum cannot be the originator. For $P_w^O$, it is the probability of being the originator of a write quorum, that is, the size of the quorum divided by the number of sites of the system.

For simplicity we assume that the number of sites are as follows: an even number for Majority, a perfect square for Grid, and the size of a complete d-ary tree for Tree. This simplification considers optimal quorum sizes for all protocols. Except for the tree quorum, the calculations are straightforward. The tree quorum is a special case in that it does not provide any obvious mechanism to automatically distribute the load. All other protocols ensure that all sites do about the same amount of work without any special arrangement. For Tree quorum, if we assume there is a single tree structure for the database, then the scalability is 1. Obviously, if the entire load goes through the root, the system will only scale as much as the root. If we assume all sites have the same capacity, then the scale out factor is simply 1 for all values of $n$, $w$, and $wo$. To avoid this limitation, we will assume that the database can be divided in $n$ partitions, each one assigned to one site. We will further assume that each site/partition is assigned a different tree where that site acts as the root of the tree of the partition. The traffic will be divided among the partitions so that transactions are executed only within one partition (otherwise, serializability cannot be guaranteed).

An additional shortcoming of the tree quorum is that the quorum size grows when failures occur. For the sake of simplicity, in our scalability analysis we will only consider the quorum size during normal operation (i.e., without failures). We provide a general expression for trees of degree $d$ (row Tree(d) in the table) and resolve the expression for ternary trees (row Tree(3) in the table).

For analysis purposes, we will work with a *d-ary* tree (with $d = 2 \cdot m - 1, m > 1$, hence $d$ is odd and larger than 3) of height $h$ and a total number of sites of $n$. Thus, $h = \lfloor log_d n \rfloor + 1$ and $n = \sum_{i=0}^{\lfloor log_m n \rfloor} d^i$ (assuming the root is at level 0, its children at level 1, ...). Read quorums are unitary and thus:

$$P_r^{tree} = \frac{1}{n}$$

A write quorum is a tree of degree $\frac{d+1}{2}$, superimposed on the original tree of degree $d$ and with the same height. The size of tree corresponding to the write quorum is given by:

For $d = 3$:

$$\sum_{i=0}^{\lfloor log_3 n \rfloor} 2^i = 2^{1+\lfloor log_3 n \rfloor} - 1 \sim 2 \cdot 2^{\frac{log_2 n}{log_2 3}} \sim 2 \cdot n^{\frac{1}{log_2 3}} \sim 2 \cdot n^{0.63}$$

For $d > 3$:

$$\sum_{i=0}^{\lfloor log_d n \rfloor} \binom{d+1}{2}^i = \frac{(d+1) \cdot \frac{d+1}{2}^{\lfloor log_d n \rfloor} - 2}{d-1}$$

$$\sim \frac{(d+1) \cdot n^{log_d \frac{d+1}{2}} - 2}{d-1}$$

## 4.4 Protocol comparison: scalability

Given the quorum probabilities, expressions 3 and 4 can be calculated for each of the protocols. The results are summarized in table 1.

Fig. 3 shows the scale out factor for the different protocols in a symmetric system as a function of $n$ and $w$. Fig. 4 shows the same type of graphs for asymmetric systems with $wo = 0.15$.

In a symmetric system, the protocols greatly depend on $w$, i.e., they are very sensitive to the proportion of write operations in the overall load. Majority quorum has a much smaller dependency but at the cost of very limited scalability. This is easy to explain given how Majority quorum distribute the load: each operation, whether read or write, requires half the sites. Thus, the system behaves as it would have twice the capacity of a single node. Tree and Grid quorums also depend on $w$, but both have the advantage of being directly proportional to $n$. For a square grid, the dependency is on $n^{0.5}$ while for a 3-ary tree the dependency is on $n^{0.37}$. Thus, in principle, Tree and Grid quorums should scale better than ROWAA and Majority. Fig. 3, however, gives a much more precise picture. For high update rates all protocols exhibit very poor scalability. As the proportion of read operations increases, ROWAA and Tree quorum are a much better choice, with Tree quorum being slightly better. Thus, Majority and Grid quorums seem to be an advantage only in applications with a large proportion of write operations: more than 50 % when compared with ROWAA and more than 70 % when compared with Tree quorum.

The comparison between the protocols is, unfortunately, not that straightforward. As pointed out above, except the tree, all protocols have a built-in load distribution mechanism. If all the nodes receive the same number of transactions the load will also be evenly distributed across the system. This is very easy to implement (e.g., by allocating transactions to sites in a round-robin fashion). In Tree quorum this is not the case. The assumption we have made is that access to the data partitions is evenly distributed, which is a different thing all together. The slightest deviation from an even distribution in data access will have an immediate impact on the scalability because of the bottleneck effect caused by the root of the data partition more frequently accessed. Since all databases have hot-spots, Tree quorum is rather difficult to use in practice as it would require very sophisticated dynamic data partitions. Additionally, and like with Grid quorum, the possible configurations are severely restricted. One cannot add an arbitrary number of nodes to a system since $n$ is dictated by the data structure used (e.g., we cannot always construct a grid or a tree for any value of $n$). The range of viable values for $n$ is very restricted in Tree quorums (e.g., $n = 1, 4, 13, ...$ for ternary trees). For Grid quorums, optimal grids are only obtained with square grids and, hence, $n$ must be a perfect square ($n = 4, 9, 16, ...$). Any deviation from a square grid leads to worse scalability (plus worse availability and larger communication overhead).

Fig. 3 demonstrates that scalability quickly degrades as the proportion of write operations in the system increases. A way to minimize this dependency is to use asymmetric systems. As Fig. 4 shows, introducing asymmetries for write operations does not change the nature of the scalability for each protocol but it does help to minimize the degradation caused by $w$. Interestingly, reducing the cost of remote writes makes Majority and Grid less attractive as the regions where they are better than ROWAA and Tree quorum are even smaller than before ($w$ must be higher).

Thus, the results for scalability can be summarized as follows:

*High scalability* can only be achieved with a low rate of update operations regardless of the protocol used.

*For most values of $w$,* ROWAA *and Tree* offer significantly better scalability than Majority and Grid.

*If the cost of remote writes decreases*, Majority and Grid are only interesting in terms of scalability for write only applications ($w \sim 1$).

# 5 Availability

In this section we analyze and compare the availability of the different protocols. We assume that failures are independent and that the probability of a site being up is $p$. We will assume that $p > 0.5$ since it has been shown that for $p < 0.5$ the best option is not to use replication [15]. The overall availability of the system will be referred to as $av$ and we will distinguish between the availability for read operations $av_R$ and the availability for write operations $av_W$. Thus:

$$av = w \cdot av_W + (1 - w) \cdot av_R \qquad (5)$$

|  | $so$ | $so\,(n \gg 1)$ | $so(wo)$ | $so(wo)\ (n \gg 1)$ |
|---|---|---|---|---|
| ROWAA | $\frac{n}{1+w\cdot(n-1)}$ | $\sim \frac{1}{w}$ | $\frac{n}{1+w\cdot wo\cdot(n-1)}$ | $\sim \frac{1}{w\cdot wo}$ |
| Majority | $\frac{2\cdot n}{2\cdot w+n}$ | $\sim 2$ | $\frac{2\cdot n}{2\cdot w+n\cdot(1+w\cdot(wo-1))}$ | $\sim \frac{2}{1+w\cdot(1-wo)}$ |
| Tree(3) | $\frac{n}{1+2\cdot w\cdot(2^{\lfloor log_3 n\rfloor}-1)}$ | $\sim \frac{n^{0.37}}{2\cdot w}$ | $\frac{n}{1+w\cdot wo\cdot(2\cdot 2^{\lfloor log_3 n\rfloor}-1)}$ | $\sim \frac{n^{0.37}}{2\cdot w\cdot wo}$ |
| Tree(d) | $\frac{n}{1+w\cdot\frac{d+1}{d-1}\cdot(2\cdot\frac{d+1}{2}^{\lfloor log_d n\rfloor}-1)}$ | $\sim \frac{(d-1)\cdot n^{1-log_d\frac{d+1}{2}}}{w\cdot(d+1)}$ | $\frac{n}{1+w\cdot wo\cdot\frac{d+1}{d-1}\cdot(\frac{d+1}{2}^{\lfloor log_d n\rfloor}-2)}$ | $\sim \frac{n^{1-log_d\frac{d+1}{2}}}{w\cdot wo\cdot\frac{d+1}{d-1}}$ |
| Grid | $\frac{n}{(w+1)\cdot\sqrt{n}-w}$ | $\sim \frac{\sqrt{n}}{w+1}$ | $\frac{n}{w\cdot(1-2\cdot wo)\cdot(1-\sqrt{n})+\sqrt{n}}$ | $\sim \frac{\sqrt{n}}{1-w\cdot(1-2\cdot wo)}$ |

Table 1: Scalability of different quorums

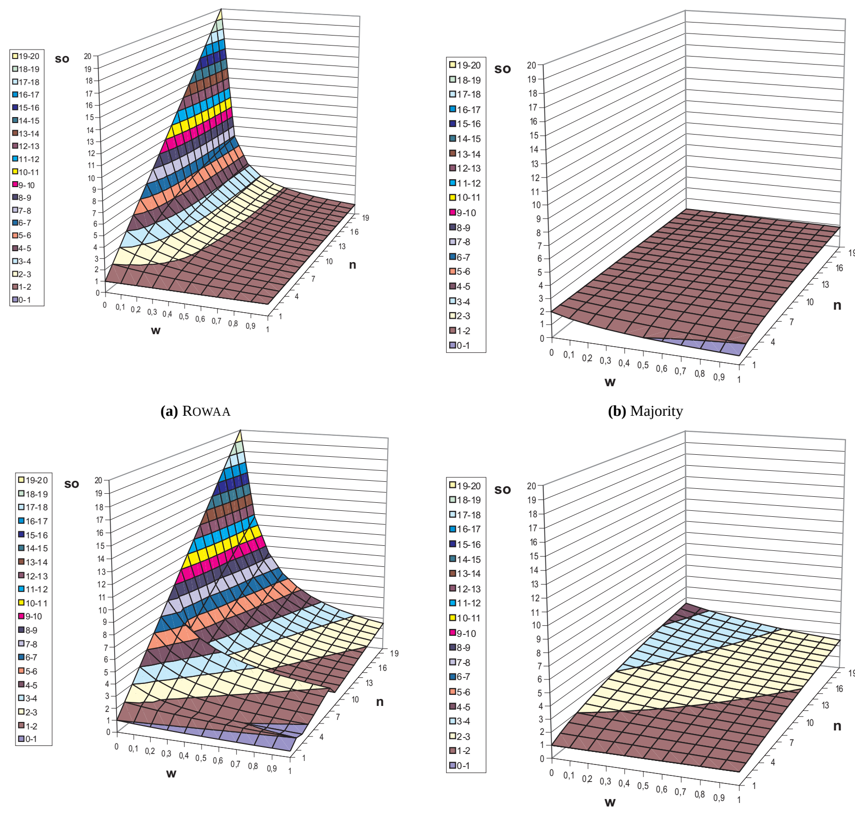


Figure 3: Scale-out for different values of $w$ and $n$

## ROWAA

Using the ROWAA protocol the system is available for both write and read as long as one site is available. That is, it tolerates up to $n-1$ failures. The probability that all sites fail is: $(1-p)^n$. Therefore, the availability ROWAA is given by:

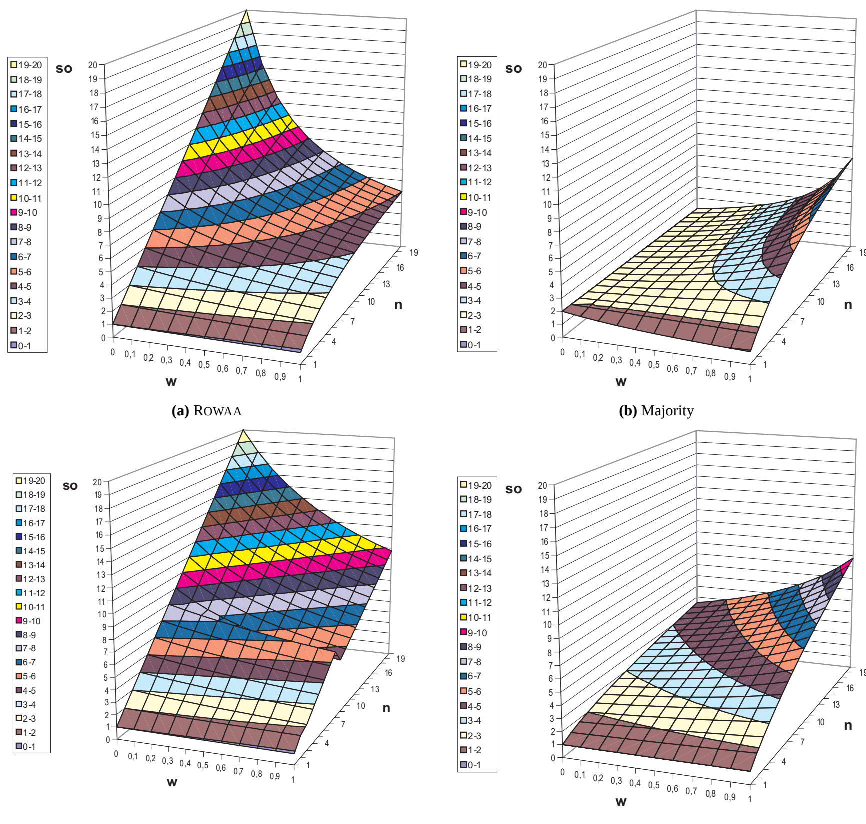
$$av = av_R + av_W = 1-(1-p)^n$$

## Majority

In a majority quorum protocol, the system can progress as long as there is a majority of available sites. For simplicity in the notation, we assume an odd number of sites, $n = 2k+1$ (that is, both write and read quorums require $k+1$ sites and we do not need to distinguish between the two types of quorums). From here, the availability is given by:

$$av = Probability(k+1\ copies\ up) + Probability(k+2\ copies\ up) + \ldots + Probability(n\ copies\ up) =$$

$$\sum_{i=1}^{k+1}\binom{n}{k+i}p^{k+i}(1-p)^{k+1-i}$$



**(a)** ROWAA **(b)** Majority **(c)** Tree(3) **(d)** Grid

Figure 4: Scale-out for different values of $w$ and $n$ for $wo = 0.15$

## Tree

We assume a tree of degree $d$, where $d$ is odd and $d > 3$, i.e., $\frac{d+1}{2}$ builds a majority of children. The read availability, $av_{r_h}$, of a tree of height $h$ and degree $d$ is characterized by the following recurrence relations [1]:

$$av_{r_{h+1}} = Probability(root\,is\,up) + Probability(root\,is\,down)$$
$$\cdot Probability(a\ majority\ of\ subtrees\,is\,read\ available)$$

That is:

$$av_{r_{h+1}} =$$
$$p + (1-p) \cdot \sum_{i=0}^{\frac{d+1}{2}-1} \binom{d}{\frac{d+1}{2}+i} \cdot av_{r_h}^{\frac{d+1}{2}+i} (1 - av_{r_h})^{\frac{d+1}{2}-1-i}$$
$$av_{r_0} = p$$

The corresponding recurrence relations for write quorums are:

$$av_{w_{h+1}} = Probability(root\,is\,up)$$
$$\cdot Probability(a\ majority\ of\ subtrees\,is\,write\ available)$$

That is:

$$av_{w_{h+1}} = p \cdot \sum_{i=0}^{\frac{d+1}{2}-1} \binom{d}{\frac{d+1}{2}+i} \cdot av_h^{\frac{d+1}{2}+i} (1 - av_h)^{\frac{d+1}{2}-1-i}$$

$$av_{w_0} = p$$

## Grid

The availability of a grid does not only depend on the number of elements in the grid, but also on the grid configuration, that is, the number of rows and columns. Following [7] the write availability of a grid of size $r \times c$ is:

$$av_W = (1 - (1-p)^r)^c - (1 - p^r - (1-p)^r)^c$$

Similarly, the read availability of a grid of size $r \times c$ is:

$$av_R = (1 - (1-p)^r)^c$$

If, as before, we assume a square grid ($r = c = \sqrt{n}$), the overall availability using equation 5 is given by:

$$av = (1 - (1-p)^{\sqrt{n}})^{\sqrt{n}} - w \cdot (1 - p^{\sqrt{n}} - (1-p)^{\sqrt{n}})^{\sqrt{n}}$$

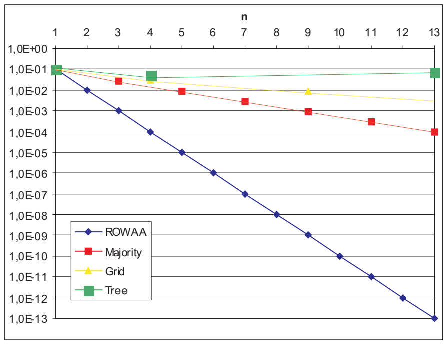


Figure 5: Unavailability for all protocols ($p = 0.9$)

### 5.1 Protocol comparison: availability

For comparison purposes, we will work with the *unavailability* of the system. The unavailability is calculated as $1 - av$ and represented in logarithmic scale. Thus, an unavailability of $10^{-i}$ corresponds to an availability of $i$ nines (e.g., two nines is 0.99). Fig. 5 compares the different protocols in terms of availability. To make it a fair comparison, we will only consider optimal configurations for each protocol. For Majority quorum we consider only configurations with an odd number of sites; for Tree quorum we consider complete ternary trees of heights 1 to 3 (i.e., with 1, 4, and 13 sites); for Grid quorum we consider configurations corresponding to square grids.

As Fig. 5 shows, ROWAA has, by far, the best availability. Each additional site added to the system increases its availability by one nine. In comparison, Majority quorum requires 4 additional sites to increase the availability by one nine. In both protocols, the availability is independent of $w$ and directly proportional to $n$. For Grid quorum, the results are quite different. Grids of size 4, 9, and 16 are perfect squares and the availability slightly increases as $n$ grows. The increase for square grids is, nonetheless, much smaller than for ROWAA or Majority quorum. Tree quorum shows the worst availability of all protocols. In fact, the availability remains close to $0.9$ and does not improve as more sites are added to the system.

We are aware that there are alternative versions of the tree and grid protocols that improve the overall availability (e.g., [17] shows a variation of grid with an improved availability). However, all quorum based protocols have an upper bound in terms of availability. This upper bound is defined by the probability of a quorum being available. Assuming a quorum size of $qs$:

$$av \leq 1 - (1 - p)^{qs} \tag{6}$$

The effects of his upper bound have been previously studied [15] and the results are consistent with our analyses. In practice, this upper bound implies that, of all protocols based on quorums (majority, grid, and tree), majority is the most available quorum for $p > 0.5$. Tree and grid are worse than majority as they have smaller quorums. In general, the availability of Tree quorum is worse than that of the Grid quorum, but the potential availability of Tree quorum is better than the one for Grid quorum. This peculiarity has to do with the asymmetries in the tree protocol [17].

These results indicate that the best option from the availability point of view is ROWAA. It could be argued, however, than ROWAA does not tolerate network partitions while the other protocols do. However, ROWAA can be made to tolerate network partitions fairly easily without affecting its behavior by using a primary component approach where only the primary component is allowed to progress. That is, a partition is only allowed to progress if it contains at least a quorum. In fact using a primary-component approach the primary component can be defined by using any quorum system. With this in mind, the results for availability can be summarized as follows:

ROWAA *and Majority quorums* are the only protocols that provide a relevant increase in availability as more sites are added to the system. ROWAA offers linear increases (in a logarithmic scale) in availability as more sites are added, while Majority quorum requires 4 additional sites to match the increase in availability reached when one site is added to ROWAA.

*The availability of Tree and Grid quorums* decreases as the proportion of writes in the load increases (i.e., as $w$ increases). The availability of ROWAA and Majority quorum is independent of $w$.

*Tree and Grid quorums* are highly sensitive to the configuration and provide very bad availabilities for certain values of $n$. This produces a non-monotonic availability, and thus, the addition of new sites can sometimes result in a reduced availability.

## 6 Communication overhead

Replication requires the participating sites to coordinate their activities by exchanging messages. In practice, this can have a significant impact on the overall behavior of the protocol. On the one side, CPU cycles are lost in dealing with the messages (flattening, sending, receiving, and unflattening) [10]. On the other side, the network bandwidth might be exceeded, resulting in additional delays. The message overhead affects not only the scalability but also the availability [16]. In this section we study the message overhead of the different protocols.

### 6.1 Message cost

To get an accurate picture of the communication overhead, one needs to take into consideration the transactional context in which data replication occurs. Thus, we follow a slightly different model from the one used in previous studies where issues like 2PC and system asymmetries have not been taken into account. For the purposes of our study, we consider only the best possible implementation (the one with the least number of messages).

We will assume that all write operations are executed locally on a shadow copy. Thus, updates are sent to the replicas only at the end of the transaction in one single message. If a transaction contains write operations, the participating sites have to agree on the outcome of the transaction using a 2PC protocol. Sending the write operations can be combined with the *vote request* message of the 2PC protocol. The participating sites must respond with a *vote* message. In the last phase the originator sends a *commit* or *abort* message to all sites in the write quorum. In contrast to write operations, read operations cannot be delayed until the end of the transactions or executed on a local shadow copy. Hence, for each read operation of a transaction, the originator of the transaction must send a read request to each member of the read quorum and each participating site must return a reply message containing the read value and its version. For ROWAA and Tree quorum, we will assume the read is performed locally and no messages are needed for read operations.

In addition, to calculate the message overhead per operation, we have to take into consideration the number of write operations per transaction. Since the message overhead caused by write operations is constant per transaction, the message overhead per individual write operation decreases with increasing number of write operations per transaction. If transactions only have on average one write operation, then each write operation in the system causes three message rounds. If transactions have on average ten write operations, then the overhead per write operation is only a tenth. Note, that the number of write operations per transaction is not determined 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 number of message per operation is

$$msg = w \cdot \frac{3 \cdot (wq - 1)}{o_w} + (1 - w) \cdot 2 \cdot (rq - 1)$$

for point-to-point messages. If a multicast primitive is available, the number of messages exchanged changes slightly. For all updates, one message is needed for the *vote request*, $wq - 1$ messages are needed to get the *vote* message of each participant, and one more message is required to *commit* or *abort* the transaction. For each read operation, one needs a message to request the read, and $rq - 1$ messages to get the responses from the participants. With this, the average number of message per operation becomes

$$msg = w \cdot \frac{wq + 1}{o_w} + (1 - w) \cdot rq$$

## 6.2 Protocol comparison: message overhead

Table 2 shows the average number of messages per operation needed for each of the protocols, using point-to-point communication and when multicast is available. As in the rest of the paper, we provide formulas for the optimal quorum sizes of each protocol.

In read intensive environments ROWAA and the Tree protocol behave the best, while Majority has the worst behavior. Even a multicast environment does not help much. The grid protocol behaves slightly better than Majority but not by a large margin. For high update rates, ROWAA has the worst behavior in point-to-point networks. However, once multicast is available, the overhead caused by ROWAA significantly decreases and there are no significant differences between the different protocols. We can summarize as follows:

*The message overhead* is proportional to $n$ for all protocols, although the dependency is much smaller for Tree and Grid quorums than for ROWAA and Majority quorum. The message overhead is significantly reduced if multicast facilities are available.

*For read intensive operations,* the best option is ROWAA independently of whether multicast facilities are available or not. The worst option is Majority quorum.

*For write intensive applications* Tree and Grid quorums create the least message overhead. However, if multicast facilities are available, there is no significant difference between the protocols.

## 7 Discussion and Conclusions

With the results presented so far, we can now give a quite comprehensive evaluation of the protocols from a practical point of view. Note, however, that we have a very concrete application and hardware configuration. This configuration, cluster computing, is becoming increasingly pervasive and, therefore, we believe the following summary is relevant for software and application designers working on clusters.

*Scalability:* ROWAA and Tree quorum are the correct choices for read intensive environments. For very write intensive environments (close to write only applications), Grid and Majority quorum offer better scalability.

*Availability:* ROWAA provides the best possible availability: the availability increases linearly with each additional replica added to the system. Moreover, the availability does not depend on the type of workload (read intensive or write intensive). Majority also provides reasonable availability. Grid and Tree quorums are poor choices from the point of view of availability.

*Message overhead:* Many clusters are built nowadays in a star configuration with a switch at the center. As a result, multicast is the normal mode of operation. With this in mind, ROWAA and the Tree quorum have the best behavior in terms of message overhead. Majority quorum has the worst overhead.

| protocol | point to point (a) | multicast (b) |
|---|---|---|
| ROWAA | $\frac{3 \cdot w}{o_w} \cdot (n-1)$ | $\frac{w}{o_w} \cdot (n+1)$ |
| majority | $\frac{3 \cdot w}{o_w} \cdot \frac{n-1}{2} + (1-w) \cdot (n-1)$ | $\frac{w}{o_w} \cdot \frac{n+3}{2} + (1-w) \cdot \frac{n+1}{2}$ |
| grid | $\frac{3 \cdot w}{o_w} \cdot (2 \cdot \sqrt{n} - 2) + 2 \cdot (1-w) \cdot (\sqrt{n} - 1)$ | $\frac{w}{o_w} \cdot 2 \cdot \sqrt{n} + (1-w) \cdot \sqrt{n}$ |
| tree(3) | $\frac{3 \cdot w}{o_w} \cdot (2^{1+\lfloor log_3 n \rfloor} - 2)$ | $\frac{w}{o_w} \cdot 2^{1+\lfloor log_3 n \rfloor}$ |

Table 2: Communication overhead

*Configuration:* ROWAA and Majority do not impose any restriction in the number of nodes and require very little in terms of structuring the system. Grid and the Tree quorum, are extremely restrictive. Grid quorum is not as limiting but deviations from square grids results in significantly worse behavior. Tree quorum can be used in very few configuration and requires a significant effort to structure the system.

*Realistic loads:* Typical workloads are neither extremely read intensive nor extremely write intensive. In most cases, there is a tendency to have more reads than writes (with a 70/20 or even 80/20 ratio in most cases). Thus, most applications greatly benefit from local reads. This makes ROWAA and Tree quorum better choices for systems that must support a wide range of loads.

*Load balancing:* A key aspect in the performance of a cluster is load balancing. In this ROWAA and Majority quorum excel. Grid quorum is also adequate but it is very depedent on the configuration chosen (particularly if it is not a square). Tree quorum is almost not an option from this point of view. Although it would be theoretically possible to balance the load using Tree quorum, in practice it almost impossible and would require to change the system configuration every time the load changes.

The conclusion we draw from these results is that ROWAA is the best choice for a wide range of applications over clusters. It offers good scalability (within the limitations of replication protocols), very good availability, and given the networks used in most clusters, a reduced communication overhead. It also has the significant advantage of being very simple to implement. For very peculiar loads and configurations, it is possible that some variation of quorum does better than ROWAA. The analyses provided in the paper clearly identify these situations and can serve as a guide to system designers for the few cases in which ROWAA is not adequate.

## References

[1] D. Agrawal and A. E. Abbadi. The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. In *Proc. Of the 16th VLDB Conf.*, Brisbane, Australia, 1990.

[2] M. Ahamad and M. H. Ammar. Performance Characterization of Quorum-Consensus Algorithms for Replicated Data. *IEEE TSE*, 15(4):492–496, Apr. 1989.

[3] G. Alonso, W. Bausch, C. Pautasso, M. Hallett, and A. Kahn. Dependable Computing in Virtual Laboratories: Logging and Recovery of Long Lived Computations. In *IEEE Int. Conf. in Data Engineering*, 2001.

[4] T. Anderson, Y. Breitbart, H. F. Korth, and A. Wool. Replication, Consistency, and Practicality: Are These Mutually Exclusive? In *ACM SIGMOD Conference*, 1998.

[5] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, Reading, MA, 1987.

[6] R. Buyaa(Ed.). *High Performance Cluster Computing: Architectures and Systems*. Prentice-Hall, 1999.

[7] S. Y. Cheung, M. Ahamad, and M. H. Ammar. The grid protocol: a high performance scheme for maintaining replicated data. In *Proc. of ICDE'90*, pages 438–445, 1990.

[8] D. K. Gifford. Weighted Voting for Replicated Data. *7th ACM Symp. on Operating Systems*, pages 150–162, 1979.

[9] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proc. of the SIGMOD*, pages 173–182, Montreal, 1996.

[10] R. Guerraoui, P. Felber, B. Garbinato, and K. R. Mazouni. System support for object groups. In *ACM OOPSLA'98*, Oct. 1998.

[11] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-R, A new way to implement Database Replication. In *Proc. of VLDB'01*, 2000.

[12] M. Naor and A. Wool. The Load, Capacity, and Availability of Quorum Systems. *SIAM Journal of Computing*, 27(2):423–447, Apr. 1998.

[13] M. Nicola and M. Jarke. Performance Modeling of Distributed and Replicated Databases. *IEEE Trans. on Knowledge and Data Engineering*, 12(4):645–672, July 2000.

[14] M. Patiño Martínez, R. Jiménez Peris, B. Kemme, and G. Alonso. Scalable Replication in Database Clusters. In *In Proc. of Int. Conf. on Distributed Computing, DISC'00, LNCS-1914. Toledo, Spain*, pages 315–329, 2000.

[15] D. Peleg and A. Wool. The Availability of Quorum Systems. *Information and Computation*, 123(2):210–223, 1995.

[16] D. Saha, S. Rangarajan, and S. K. Tripathi. An analysis of the average message overhead in replica control protocols. *IEEE Trans. on Paral. and Dist. Syst.*, 7(10), 1996.

[17] O. Theel and H. Pagnia. Optimal Replica Control Protocols Exhibit Symmetric Operation Availabilities. In *Proc. of Symp. on Fault-Tolerant Computing (FTCS)*, 1998.

[18] R. H. Thomas. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems*, 4(9):180–209, June 1979.