Are Quorums an Alternative for Data Replication?

RICARDO JIMÉNEZ-PERIS and M. PATIÑO-MARTÍNEZ
Universidad Politécnica de Madrid (UPM), Madrid, Spain
GUSTAVO ALONSO
Swiss Federal Institute of Technology (ETHZ), Zürich, Switzerland
and
BETTINA KEMME
McGill University, Montreal, Quebec, Canada

Data replication is playing an increasingly important role in the design of parallel information systems. In particular, the widespread use of cluster architectures often requires to replicate data for performance and availability reasons. However, maintaining the consistency of the different replicas is known to cause severe scalability problems. To address this limitation, quorums are often suggested as a way to reduce the overall overhead of replication. In this article, we analyze several quorum types in order to better understand their behavior in practice. The results obtained challenge many of the assumptions behind quorum based replication. Our evaluation indicates that the conventional read-one/write-all-available 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-available strategy is much simpler to implement and more flexible than quorum-based approaches. In this article, we show that, in addition, it is also the best choice using a number of other selection criteria.

Categories and Subject Descriptors: H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*performance evaluation*

This article is an extended version of JIMÉNEZ-PERIS, R., PATIÑO-MARTÍNEZ, M., ALONSO, G., AND KEMME, B. 2001. How to select a replication protocol according to scalability, availability, and communication overhead. In *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS)* (New Orleans, La.). IEEE Computer Society Press, Los Alamitos, Calif., pp. 24–33.
This work has been partially funded by the Spanish National Science Foundation (MCYT), contract number TIC2001-1586-C03-02.
Authors' addresses: R. Jiménez-Peris and M. Patiño-Martínez, Facultad de Informática, Universidad Politécnica de Madrid, Madrid, Spain; email: {rjimenez,mpatino}@fi.upm.es, Web: http://lsd.ls.fi.upm.es/~rjimenez; http://lsd.ls.fi.upm.es/~mpatino; G. Alonso, Department of Computer Science, Swiss Federal Institute of Technology (ETHZ), Zürich, Switzerland; email: alonso@inf.ethz.ch, Web: http://www.inf.ethz.ch/personal/alonso; B. Kemme, School of Computer Science, McGill University, Montreal, Quebec, Canada; email: kemme@cs.mcgill.ca, Web: http://www. cs.mcgill.ca/~kemme/.

General Terms: Performance, reliability

Additional Key Words and Phrases: Data replication, quorums, scalability, availability, distributed transactions.

1. INTRODUCTION

The text book approach to deal with performance and availability problems in data replication protocols is to use *quorums*. Quorums reduce the number of copies involved in reading or updating data. Hence, in theory, quorums should reduce the overall cost of replication in terms of performance penalties, communication overhead, and system's availability [Bernstein et al. 1987; Weikum and Vossen 2001]. Following this idea, many text books describe quorums as the way to, for example, deal with network partitions [Coulouris et al. 2000; Bernstein et al. 1987], minimize communication overhead [Bacon 1997], increase availability [Lewis et al. 2002], and balance the cost between read and write operations [Weikum and Vossen 2001]. Thus, from the existing literature, one could readily conclude that quorums can be used to improve the performance and availability of replication protocols.

Although the availability and load distribution of quorum systems have been studied in much detail [Rangarajan et al. 1993; Naor and Wool 1998; Amir and Wool 1996, 1998; Peleg and Wool 1995; Ahamad and Ammar 1989], existing studies make three simplifying assumptions that distort the comparisons. These three assumptions are: (a) asymptotic behavior as being representative of the overall system's behavior; (b) read and writes have the same cost regardless of whether they are local or remote; and (c) scalability does not need to be monotonic.

In terms of comparison, most existing availability studies used asymptotic analysis, for example, Naor and Wool [1998], to draw conclusions about the comparative characteristics of different quorum techniques. Unfortunately, eager replication does not scale beyond a few tens of sites. For any comparison to be practically relevant, it is necessary to take into account the constant and multiplicative factors discarded in an asymptotic analysis. Similarly, to facilitate the analysis, researchers have assumed that transaction processing is entirely symmetric across the system (e.g., Nicola and Jarke [2000] and Agrawal and Abbadi [1990b, 1990a]). This is probably one of the most distorting factors in such analyses as transaction processing in a replicated, distributed database is highly asymmetric. An example of such asymmetries are those caused by common optimization strategies like propagating tuple changes directly rather than SQL statements. Finally, quorum formation techniques are typically studied only for the best possible configuration. This hides the fact that many forms of quorums impose significant restrictions on the number of nodes in the system (e.g., Cheung et al. [1990] and Agrawal and Abbadi [1990b]), and might degenerate if used in a nonoptimal configuration. For scalability purposes, it is crucial to be able to grow the system by adding a few nodes at a time. Otherwise, it becomes very difficult to use it in a real system. Ideally, the quorum strategy should exhibit monotonic scalability.

With these important practical considerations as the basis for the analysis, we have studied and compared some of the most relevant forms of quorums. Due to the more realistic assumptions, we believe the analysis offers a comprehensive and fair picture of existing quorums and is, therefore, an important contribution by itself. Our results challenge the notion that quorums improve performance, availability, or communication overhead. The results of our analyses indicate that the conventional read-one/write-all-available approach (ROWAA) is clearly the best choice for a wide range of applications requiring data replication. In terms of scalability, ROWAA is clearly one of the best choices possible for most applications since it minimizes the read overhead. Our analyses show that most quorum systems outperform ROWAA only under extreme write ratios (80–90% of the operations are updates). In terms of availability, ROWAA is far better than any quorum, even if we consider only optimal configurations for each quorum technique. When monotonic scalability is considered, many types of quorums become infeasible and ROWAA is the only reasonable option. In terms of communication, it can be argued that ROWAA does not tolerate network partitions. However, this can be easily solved by adopting a primary partition approach. The primary partition would be the one containing at least one quorum. With this, ROWAA provides the same availability as quorums and, as our analyses show, has a much better behavior than any quorum system when scalability, availability, and communication overhead are considered as a whole. This is an important result since the read-one/write-all strategy is much simpler to implement and more flexible than quorum based approaches.

The article is organized as follows. Section 2 presents the system model and some definitions. Quorum protocols are studied according to their structure. Section 3 presents the ROWAA protocol and voting quorums. Different quorum systems based on grids are studied in Section 4. Finally tree quorums are described in Section 5. Section 6 compares the best protocol of each of these three sections. Related work is presented in Section 7 and conclusions in Section 8.

2. SYSTEM MODEL AND DEFINITIONS

In this section, we first introduce our transaction and quorum model. Then, we describe three performance measurements: scalability and how work is distributed in the system, availability, and communication overhead. We would like to mention that the analysis ignores some aspects of the problem. One of them is concurrency control and its effects on transactional conflicts, response time delay, abort rate, etc. For such analysis we refer, for example, to Gray et al. [1996], Carey and Livny [1988], and Yu et al. [1993]. A second issue is the overhead of forming and maintaining quorums, which can be considerable in some cases. Thus, our measurements will produce results that should be seen as an upper bound of what can *potentially* be achieved since including these additional costs will only decrease the performance.

2.1 Model

A replicated database consists of a group of sites $N = \{N_1, N_2, \ldots, N_n\}$ that communicates by exchanging messages. Sites are fail-stop, and site failures can be detected. We consider a crash-recovery model where sites can recover and rejoin the system after synchronizing their state with 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.

Clients interact with the database by issuing transactions. Transactions are executed atomically, that is, a transaction either commits or aborts. Transactions are partially ordered sets of read (r) and write (w) operations. Two transactions conflict if they access the same data item and at least one of them is a write operation. If a transaction contains write operations, a 2-phase-commit protocol (2PC) at the end of the transaction is executed among all sites. We assume replica control is combined with a concurrency control method such that one-copy-serializability [Bernstein et al. 1987] is guaranteed: each copy must appear as a single logical copy and the execution of concurrent transactions over the physical copies is equivalent to the serial execution over the logical copy.

A client submits a transaction—and with it all the operations of this transaction—to any of the sites in the system. This site is called the originator of the transaction and its operations, and coordinates with the rest of the system. A transaction and its operations are called local at the site it is submitted to, and remote at the other sites.

We consider in the study two kinds of transactions: queries, which contain only read operations, and update transactions.

The number of sites where a read or write operation must be performed is a *quorum*. In particular, read and write quorums must be such that read and write operations or two write operations on the same data item overlap (they access at least one common copy). That is, any read quorum must overlap with any write quorum, and any two write quorums must overlap between them. The following definitions present these properties formally. Some of them are borrowed from Naor and Wool [1998].

Definition 1 (*Quorum System*). A set system $S = \{S_1, S_2, \ldots, S_n\}$ is a collection of subsets $S_i \subseteq N$ of a finite universe N. A quorum system defined over a set of sites N is a set system S that has the property: $\forall i, j \in \{1 \cdots n\},\ S_i \cap S_j \neq \emptyset$

The nonempty intersection property is crucial in that it allows any quorum to make decisions on behalf of the whole system, and still guarantee overall consistency.

Definition 2 (*Write Quorum*). Given a quorum system S, each $S_i \in S$ is a write quorum.

The cardinality of a set S is denoted by $|S|$.

Definition 3 (*Degree of an Element*). The degree of an element $x \in N$ in a set system S is defined as the number of sets that contain x: $deg(x) = |\{S_i \in S,\ x \in S_i\}|$.

Definition 4 (*s-Uniformity*). A set system S is s-uniform if $\forall S_i \in S,\ |S_i| = s$.

s-uniformity indicates that all sets are of the same size and that the size is s. This property is important, since load distribution among the sites is directly related to s-uniformity. Systems that are not s-uniform have problems with load balancing and, therefore, with scalability.

Definition 5 (*(s, d)-Fairness*). A set system S is (s, d)-fair if it is s-uniform and $\forall x \in N,\ deg(x) = d$.

(s, d)-fairness adds an additional dimension to s-uniformity. With s-uniformity, all sets must be of the same size. In an (s, d)-fair system, all sets must be of the same size and all sites must equally participate in all sets (all sites must have the same degree). (s, d)-fairness expresses the symmetry of a quorum system, which impacts on the system scalability and reliability.

Definition 6 (*Read-Write Quorum*). A read-write quorum system, over the set of sites N, is a pair (R, W) where W is a quorum system, and R a set system with the following property: $\forall W_i \in W,\ \forall R_i \in R,\ W_i \cap R_i \neq \emptyset$.

Definition 7 (*Read Quorum*). Given a read-write quorum system (R, W), each $R_i \in R$ is called a read quorum.

2.2 Scalability

In order to measure the scalability of a replicated system we will consider a read-write quorum (R, W) where both R and W are (s, d)-fair. That is, all sites have the same probability to belong to a read (write) quorum and read (write) quorums having the same size. We denote the size of a write quorum with wq, and the size of read quorums with rq. From here, $P_w = \frac{wq}{n}$ is the probability for a site to participate in a write quorum, and $P_r = \frac{rq}{n}$ the probability for a site to participate in a read quorum. From here, we derive the following definitions:

Definition 8 (*Symmetry in a Quorum System*). A quorum system is considered symmetric if all operations have the same cost, regardless of whether they are local or remote operations.

Definition 9 (*Asymmetry in a Quorum System*). A quorum system is considered asymmetric if some operations have different cost, depending on whether they are local or remote operations.

Let L be the total processing capacity of the system, that is, the maximum number of operations per time unit that can be handled by 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.

Definition 10 (*Load at a Site in a Symmetric System*). Let t be the processing capacity of one site, that is, the number of operations each site is able to execute per time unit, t and L are related in the following way:

$$t = P_w \cdot L_w + P_r \cdot L_r, \tag{1}$$

which in terms of L can be rewritten as follows.

$$t = L \cdot (w \cdot P_w + (1 - w) \cdot P_r) \tag{2}$$

Definition 11 (*Symmetric Scale Out Factor, so*). The scalability (scale out factor) of a system is given by the total processing capacity of the entire system (L) divided by the processing capacity of one site (t).

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

Expressions similar to (3) can be found in Wool [1998], Kemme [2000], and Jiménez-Peris et al. [2001]. However, distributed databases are rarely symmetric. Instead, many commercial database systems perform asymmetric update operations. An SQL statement like `update employee set salary = salary + 1000 where level = 1` is parsed, optimized and executed at the local site. Most of the execution time is spent scanning the entire `employee` table for relevant records; only a handful of records might be updated. In order to avoid redundant work, the remote sites do not reexecute the SQL statement. Instead, the local site sends to all other sites the key to the employee records that need to be updated along with the new salary values. Remote sites directly access and update the relevant records (through special index structures) without scanning the entire table. Experiments in Kemme and Alonso [2000] and Jiménez-Peris et al. [2002] have shown that such an approach can reduce the execution costs at remote sites by a factor of 5.

In theory, this strategy can be applied to both read and write operations. However, if we look at, for example, relational database systems, we can see a significant difference between read and write operations. A write operation usually writes one or more records of a single table (SQL `update`, `delete`, and `insert` statements). As a result, if a write quorum does not include all sites in the system, each site might have up-to-date records (with the highest version number) and stale records (not having the highest version number) in a single table. In comparison, `select` statements are usually more complex than write statements, often involving transformations of the original tables (e.g., n-way joins). A single query can scan thousands of records and return hundreds of records or a single aggregation value to the user. For each of the records to be scanned, one has to determine the latest version. Since there might not exist a single site in the system that has the latest version of each record, each site has to first collect all records relevant for the query, then the latest version for each of these records has to be determined, and finally the result records or the aggregation value has to be calculated. In summary, since scanning the records is the most intensive work, and this work has to be done by all sites, read operations are essentially always symmetric.

Hence, in here, we 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:

$$P_w = P_w^O + P_w^R, \tag{4}$$

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

Definition 12 (*Asymmetric Scale Out Factor, so_{wo}*). The scale out factor (so_{wo}) in a asymmetric system is defined by the following.

$$so_{wo} = \frac{L}{t} = \frac{1}{w \cdot P_w^O + w \cdot wo \cdot P_w^R + (1-w) \cdot P_r} \tag{5}$$

2.3 Availability

We assume that site failures are independent and that the probability of a site being up is p. This probability is calculated as $p = \frac{T_U}{T_U + T_D}$, where T_U is the uptime and T_D the downtime. The downtime includes the time to recover the system after the failure. In our study, we do not consider that different quorum systems might have different recovery overhead since too little information is available to differentiate the costs. We assume that $p > 0.5$, since it has been shown that for $p < 0.5$ the best option is not to use replication [Peleg and Wool 1995]. 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.

Definition 13 (*Availability of an Operation*). The availability of a read (write) operation, av_r (av_w), is the probability that a read (write) operation can be performed in the system.

Definition 14 (*System Availability*). The availability of a system (av) is the probability that an operation can be performed in the system.

$$av = w \cdot av_w + (1-w) \cdot av_r \tag{6}$$

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). The read, write, and system availabilities of a single site system are all p and the corresponding unavailability is $1 - p$.

Most of the literature proposing new quorum systems already provide detailed availability analysis. There, the term availability is often equivalent to our definition of write operation availability. Whenever possible, we take the results provided and only derive read operation and system availability.

2.4 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). On the other side, the network bandwidth might be exceeded, resulting in additional delays. The message overhead affects not only scalability but also availability [Saha et al. 1996].

Our analysis focuses on the number of messages exchanged to measure the impact on CPU time (message overhead at sending and receiving sites) and network load. Although an important factor, we do not consider the size of the messages because this information is strongly application dependent in most situations. Read and write messages in a symmetric system are SQL statements, hence, they will typically always be small. The return messages for read requests, however, might contain between a few and thousands of values depending on the application. The size of write messages in an asymmetric system is also difficult to predict. Typically, these messages will contain a few updated records, however, in special applications, they might also be quite large. As a result, we do not believe that message size can be captured without knowing the application domain.

Most of the literature proposing quorum protocols provide some form of communication analysis. However, the models and assumptions in each of these papers are often very different making it difficult to compare the different approaches. In our analysis, we provide a single model for all protocols allowing for a thorough comparison. In particular, we take into consideration issues like 2PC and system asymmetries. 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 read-write quorum systems (R, W) in which R is 1-uniform (i.e., all read quorums have size $\mathrm{rq} = 1$), we will assume that the read is performed locally and no messages are needed for read operations. Furthermore, transactions that only consist of read operations do not require any message overhead.

In addition to calculating the message overhead per operation, we have to take into consideration the number of write operations per transaction. Since the number of messages per update transaction is constant and independent of the number of write operations within the transaction,[1] the message overhead per individual write operation decreases with increasing number of write

[1]We assume that all the write operations within a transaction can be sent in a single message. The expressions in this section can be easily extended if this assumption does not hold.

operations per transaction. If transactions only have on average one write operation then each write operation in the system causes three message rounds (vote request, response, decision). 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 message overhead for point-to-point messages is defined by:

Definition 15 (*Message Overhead, Point-to-Point*). The message overhead per operation performed in a quorum system using point-to-point communication is given by the expression:

$$msg = w \cdot \frac{3 \cdot (wq - 1)}{o_w} + (1 - w) \cdot 2 \cdot (rq - 1) \tag{7}$$

If a multicast primitive is available, the number of exchanged messages varies. 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 are 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. Thus:

Definition 16 (*Message Overhead, Multicast*). The message overhead per operation performed in a quorum system using multicast communication is given by the expression:

$$msg = w \cdot \frac{wq + 1}{o_w} + (1 - w) \cdot rq \tag{8}$$

3. READ-ONE WRITE-ALL AND VOTING

3.1 Read-one Write-all

In the read-one write-all (ROWA) approach [Bernstein et al. 1987], reads are performed at a single site, while write operations must be performed at all sites. Read operations can be executed locally to minimize the communication cost.

A naive version of ROWA would require all replicas to be available to perform a write operation [Bernstein et al. 1987]. A variation of this technique known as read-one write-all-available (ROWAA) improves the availability of the database [Bernstein et al. 1980, 1987; Rothnie et al. 1980]. Although the original protocol does not tolerate partitions, the basic protocol can be extended with a primary component (partitions) approach that makes it partition tolerant.

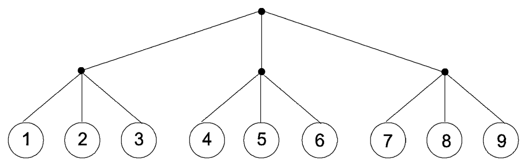

Fig. 1. Example of HQC: 9 copies organized into a 2-level hierarchy.

3.2 Voting

The MAJORITY quorum (also known as quorum consensus) [Thomas 1979] and weighted voting [Gifford 1979] use voting to define the quorums. Each site has a nonnegative number of (weights) votes. Quorums are defined so that the number of votes exceeds half of the total votes (majority). Read and write quorums must fulfill the following constraints: $2 \cdot wq > n$ and $rq + wq > n$, being n the number of sites. The minimum quorum sizes satisfying these constraints are: $2 \cdot wq = n + 1$ and $rq + wq = n + 1$ and therefore, $wq = \lfloor \frac{n}{2} \rfloor + 1$ and $rq = \lceil \frac{n}{2} \rceil = \lfloor \frac{n+1}{2} \rfloor$. Thus, a write quorum can be formed by any majority, and a read quorum by half of the system sites, if n is even, or by a majority if n is odd.

Those quorums are static, that is, the system size (total weight) is predefined. The majority can be defined dynamically, according to the current system size in order to be able to cope with failures [Eager and Sevcik 1983; Herlihy 1987; Barbará et al. 1989; Jajodia and Mutchler 1990].

Hierarchical quorum consensus (HQC) is a generalization of MAJORITY [Kumar 1991]. This generalization consists in organizing the sites into a hierarchy. This hierarchy can be represented as a complete tree where physical sites appear at the leaves of the tree. At each level i (beyond the root level) of the tree, a majority of tree nodes must be taken. For instance, nine sites can be organized into a ternary tree of three levels (Figure 1). Level 0 is the root, and level 2 is the leaf-level. To perform a read or write operation a majority of nodes at level 1 must be chosen (two out of the three nodes). For each node chosen at level 1, a majority of nodes at level 2 must be chosen (two out of three physical sites). For the example of Figure 1, the (read or write) quorum size is 4, whilst for MAJORITY is 5. The quorum size for HQC is minimal when the tree is ternary [Kumar 1991], in such a case the read and write quorum sizes are $n^{0.63}$.

Several variations of voting have been proposed like voting with witnesses [Paris 1986; Paris and Long 1991], voting with bystanders [Paris 1989], and voting with ghosts [van Renesse and Tanenbaum 1988]. The first variation introduces *witnesses* to reduce storage costs and minimize the number of replicas to achieve consistency. Witnesses are lightweight replicas that only hold version numbers. Witnesses participate as normal replicas in read and write quorums. Any quorum needs at least one conventional replica. The second and third variations are protocols that extend voting to deal with a special kind of partition failures, where only failures of gateways connecting network segments are considered (i.e., a network segment does not partition).

Multidimensional voting [Cheung et al. 1991] is a general technique to represent quorum systems that generalizes weighted voting. It assigns to each site a vector of dimension k of nonnegative weights. The weights of the n sites

Voting prot.	rq	wq	P_r	P_w^O	P_w^R
ROWAA	1	n	$\frac{1}{n}$	$\frac{1}{n}$	$\frac{n-1}{n}$
MAJORITY	$\lfloor\frac{n+1}{2}\rfloor$	$\lfloor\frac{n}{2}\rfloor + 1$	$\frac{1}{2}$	$\frac{1}{n}$	$\frac{1}{2}$
HQC	$n^{0.63}$	$n^{0.63}$	$n^{0.37}$	$\frac{1}{n}$	$\frac{n^{0.63}-1}{n}$

Fig. 2. Quorum sizes and quorum participation probabilities for selected voting protocols.

Voting prot.	so	so(wo)
ROWAA	$\frac{n}{1+w\cdot(n-1)}$	$\frac{n}{1+w\cdot wo\cdot(n-1)}$
MAJORITY	$\frac{2\cdot n}{2\cdot w+n}$	$\frac{2\cdot n}{2\cdot w+n\cdot(1+w\cdot(wo-1))}$
HQC	$n^{0.37}$	$\frac{n}{w\cdot(1+wo)+n^{0.63}\cdot(w\cdot wo-w+1)}$

Fig. 3. Scale out factors for selected voting protocols.

form a matrix of weights, md, of dimension $n \times k$. A quorum is a k-dimensional vector of non-negative values. The quorum must be satisfied for a number of dimensions l, such that $1 \leq l \leq k$. In multidimensional voting, a quorum is formed hierarchically. (1) At the vote level, the number of votes for a dimension must be greater or equal to the quorum requirement for that dimension. (2) At the dimension level, quorums must be obtained for at least l dimensions. Multidimensional voting with the quorum requirement of l out of k dimensions is termed $MD-(l,k)-voting$. The extra column gives the flexibility to represent several quorum schemes like quorum consensus and HQC.

Weighted voting and voting variations, with the exception of multidimensional voting, perform as MAJORITY in the best case [Naor and Wool 1998]. Weighted voting is useful for systems where the failure probability is not uniform. The overall system availability can improve by assigning more votes to sites with higher availability. Since, only systems with uniform failure probability are considered in our study, we will only analyze the MAJORITY and HQC protocols.

3.3 Scalability

The probability P_r of being in read quorum and the probability P_w of being in a write quorum for the selected voting protocols can be computed from the quorum sizes (Figure 2). For write operations we distinguish between being the originator of the operation (P_w^O) or just a quorum participant (P_w^R). From these probabilities the scalability expressions (3) and (5) in Section 2.2 can be calculated for each of the protocols. Figure 3 summarizes these results.[2]

For simplicity, we have linearly interpolated between the optimal configurations for each quorum type (even number of nodes in MAJORITY, number of nodes in HQC results in complete tree where each path from root to leaf has the same length). Thus, the curves can be seen as an upper bound for the scale out

[2]We assume n is even for MAJORITY.

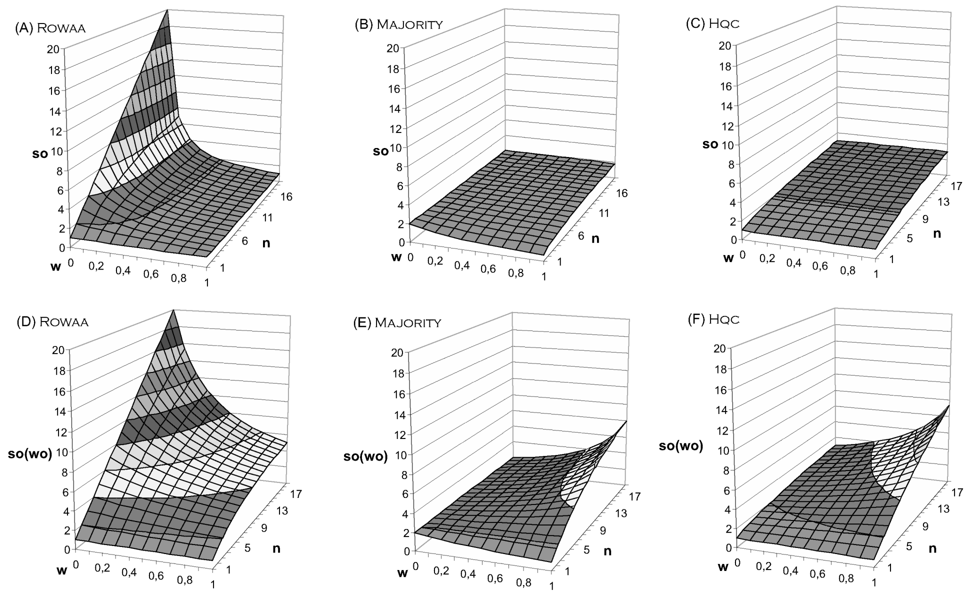

Fig. 4. Scalability (voting): (A-C) symmetric and (E-F) asymmetric load ($wo = 0.15$).

factor because scalability might be lower for nonoptimal configurations [Naor and Wool 1998].

Figure 4 shows the scale out as a function of the number of nodes and the fractions of updates in the load. In a symmetric system, voting protocols exhibit a poor scalability as can be seen in Figure 4 (A)-(C). ROWAA scales linearly for a 100% reads profile because read operations are local. But the scalability of ROWAA drops very fast when the proportion of writes increases. On the other hand, MAJORITY and HQC present an almost null scalability over the entire range with HQC performing slighly better than MAJORITY.

For asymmetric systems, the scalability of voting protocols improves noticeably as shown in Figure 4 (E)-(F). ROWAA presents good scalability when there are more reads than writes (Figure 4(D)). The drop in the scalability of ROWAA in the worst case scenario ($w = 1$) is not as sharp as in a symmetric system. MAJORITY and HQC are significant worse than ROWAA at high read rates, and outperform ROWAA at high update rates. In contrast to symmetric systems, both MAJORITY and HQC are affected by both the number of sites and the proportion of writes in an asymmetric system. HQC behaves slightly better than MAJORITY.

3.4 Availability

If there are no network partitions, a system using ROWAA is available for both write and read operations as long as one site is available. That is, it tolerates up to $n - 1$ site failures. The probability that all sites fail is: $(1 - p)^n$. Therefore, the availability ROWAA is given by:

$$av = av_r + av_w = 1 - (1 - p)^n$$

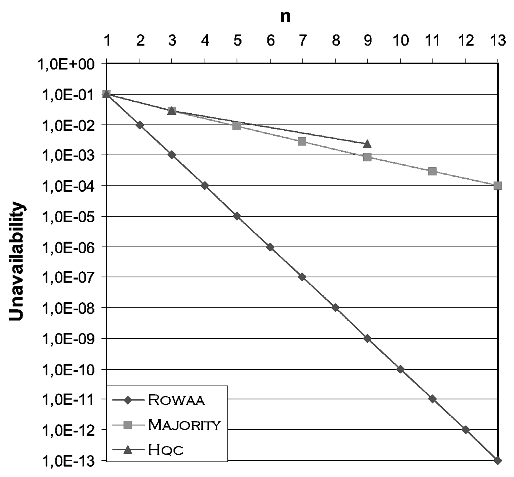

Fig. 5. Unavailability of voting protocols for $p = 0.9$.

If a primary component is considered, Rowaa exhibits the same availability as Majority. In Majority, 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, $n = 2k + 1$ (i.e., both write and read quorums require $k + 1$ sites and we do not need to distinguish between the two types of operations). From here, the system availability is given by the following expression:

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

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

The availability of Hqc is computed upwards from the leaves (av_h) of the tree (level h) to the root (level 0), where the availability of the root (av_0) is the system availability. The availability of level i (av_i) is the probability of having a majority of nodes of level $i+1$. Assuming that p is the availability of a single site, the availability formulas are recursively defined as follows [Kumar 1991]:

$$av_h = p \quad av_i = \sum_{j=2}^{3} \binom{3}{j} (av_{i+1})^j \cdot (1 - av_{i+1})^{3-j} \quad av = av_0$$

All these protocols are (s, d)-fair. Thus, all sites play the same role, they manifest the best possible availabilities for their quorum sizes. Figure 5 shows the unavailability of voting protocols as a function of the system size for $p = 0.9$.

Ignoring network partitions, Rowaa has the best availability. Considering network partitions, the most available quorum system is (minimal) Majority under the assumptions of uniform failure probability and $p > \frac{1}{2}$ [Barbará and

Voting prot.	Point-to-Point	Multicast
ROWAA	$\frac{3 \cdot w}{o_w} \cdot (n-1)$	$\frac{w}{o_w} \cdot (n+1)$
MAJORITY	$\frac{3 \cdot w}{o_w} \cdot \frac{n}{2} + (1-w) \cdot (n-2)$	$\frac{w}{o_w} \cdot \frac{n+4}{2} + (1-w) \cdot \frac{n}{2}$
HQC	$(n^{0.63}-1) \cdot (\frac{3 \cdot w}{o_w} + 2 \cdot (1-w))$	$w \cdot \frac{n^{0.63}+1}{o_w} + (1-w) \cdot n^{0.63}$

Fig. 6. Communication overhead for voting quorums.

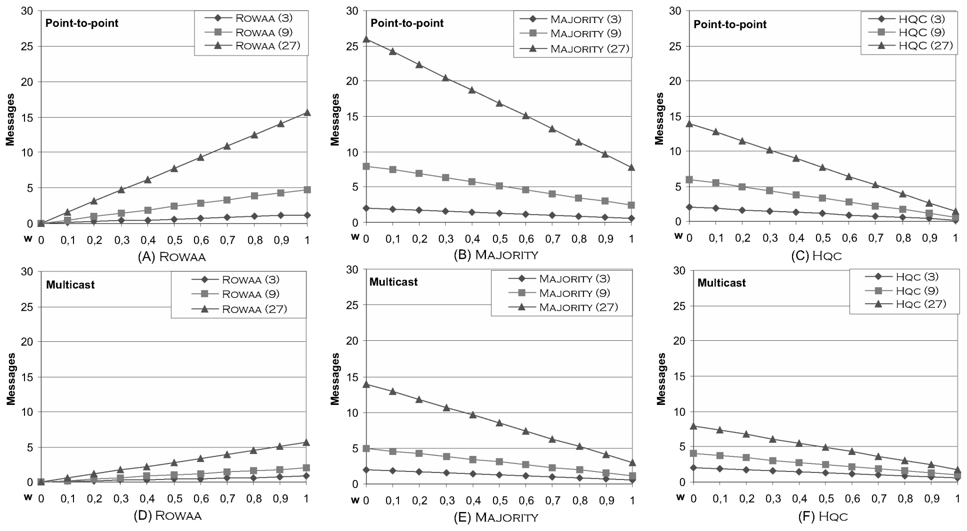

Fig. 7. Communication overhead (voting quorums and $o_w = 5$): (A-C) point-to-point communication; (D-F) multicast. The numbers in parenthesis indicate the number of sites.

Garcia-Molina 1987; Peleg and Wool 1995]. This means that the availability of MAJORITY is the upper bound for the availability of any quorum system under the previous conditions. HQC has an availability close to the one of MAJORITY when site availabilities are over 0.8.

3.5 Communication Overhead

The communication overhead is computed as the number of messages required per operation. Applying expressions (7) and (8) from Section 2.4 we obtain the overheads shown in Figure 6.

Figure 7 shows that HQC outperforms MAJORITY for both point-to-point communication and multicast, no matter the proportion of read and write operations due to the smaller quorum sizes. With point-to-point communication, ROWAA behaves better than HQC when the there are more reads than writes. With multicast, ROWAA outperforms HQC even for configurations with more writes (70%) than reads.

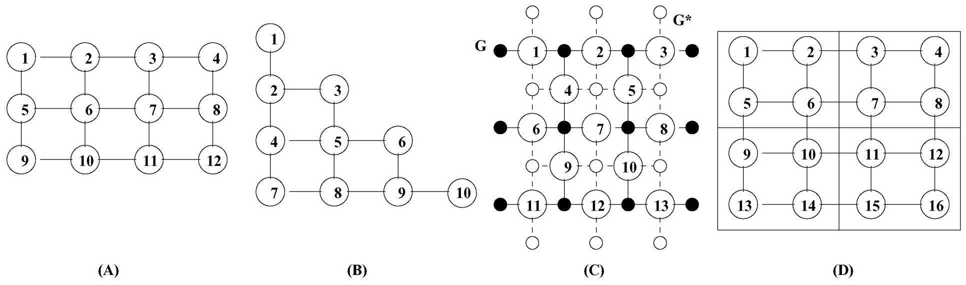

Fig. 8. Different types of grids: (A) 3 × 4 Rectangular; (B) Triangular; (C) Paths; (D) 2-level, 2 × 2 Hierarchical.

3.6 Summary of ROWAA and Voting Quorums

Overall, ROWAA seems to be the best choice among this group of protocols. It behaves better in terms of scalability and communication overhead when the proportion of reads is greater than the one of writes. If multicast is available, ROWAA produces a smaller communication overhead even when there are more writes than reads. If network partitions are ignored, ROWAA also exhibits the best availability. Extending ROWAA to cope with network partitions will bring its availability down to that of MAJORITY. Both MAJORITY and ROWAA allow easy reconfiguration in case of failures or the addition of new nodes into the system.

4. GRID QUORUMS

4.1 Basic Grid Quorum

The simplest form of grid quorum is the rectangular grid [Cheung et al. 1990]. A rectangular grid quorum organizes the n sites in a grid of r rows and c columns (i.e., $n = r \cdot c$). A read quorum consists of accessing an element of each column of the grid ($rq = c$). A write quorum requires an entire column and one element from each of the remaining columns ($wq = r + c - 1$). Given the grid in Figure 8(A), 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}.

A rectangular grid becomes optimal in terms of quorum sizes when the grid is a square. In this case, $rq = \sqrt{n}$ and $wq = 2 \cdot \sqrt{n} - 1$. In what follows we will refer to this type of grid as SQUARE.

4.2 Alternative Grid Patterns

A simple extension of the original rectangular grid is proposed by Kumar et al. [1993] by allowing read quorums to either contain one element from each column or all elements from a single column. This increases the availability of the grid without affecting the scalability (in particular, when the grid is a square read quorums remain s-uniform). The authors also propose configurations for incomplete grids. They allow not only for more flexibility in the grid structure but also exhibit better availability.

Sites in a *triangular* grid or TRIANGLE [Erdös and Lovász 1975; Lovász 1973] are arranged in d rows such that row i ($i > 1$) has i elements (Figure 8(B)). A write quorum is the union of one complete row and one element from every row below the full row. Therefore, the quorum size is always d. A read quorum can be either one element from each row or a write quorum. In the former case, all reads go through the site in the first row, therefore, we will consider that read and write quorums are formed in the same way. Since d is the number of rows, then the number of sites in the system should be $n = \sum_{i=1}^{d} i = \frac{d \cdot (d+1)}{2}$ and therefore, $d = \frac{-1+\sqrt{1+8 \cdot n}}{2}$. It should be noted that TRIANGLE is a uniform quorum system but is not (s, d)-fair. Sites at the bottom rows have a lower degree than those at the top. We will assume that there is a mechanism to distribute the load so that all sites have approximately the same load. For instance, the database can be divided in p partitions and there is a TRIANGLE for each partition. Transactions are executed within one partition.

Crumbling walls (CW) generalize TRIANGLE by not requiring row i to have i elements [Peleg and Wool 1997]. Quorums in a CW follow the same rules as in TRIANGLE. Although there are no restrictions on the number of elements of each row, a CW is a non-dominated coterie (the most available quorums) if the width of the first row is one and the width of the rest of the rows is equal or greater than two. Peleg and Wool [1997] also show that the availability of a wall improves when the row widths increase monotonically. All these requirements are met by TRIANGLE. The main advantage of CW is that logarithmic row width growth walls (CWlog) have a Condorcet failure probability[3] when the number of sites tends to infinity. However, since we are interested in realistic configurations (up to few tens of sites) and quorum sizes in CWlog are not uniform, we will restrict our study to TRIANGLE.

Lattice grids aim at increasing the availability of rectangular grids. They require quorums to be formed using *paths* from top to bottom (*vertical paths*) and from left to right (*horizontal paths*) [Theel and Pagnia-Koch 1995; Wu and Belford 1992]. A read quorum is a horizontal path while a write quorum is a horizontal and a vertical path. This idea can be applied to grids of different shape and can be used in a hierarchical manner. Lattices were further generalized by Naor and Wool [1998] who proposed several alternative grid quorums: *Paths*, *SC-Grid* and *B-Grid*. Naor and Wool already indicate that PATHS is the best option of the three so we will ignore the other two here. PATHS is based on two superimposed square grids one of size d and the other of size $d + 1$ (Figure 8(C), with $d = 2$). The number of nodes in the system, n, must be such that $n = 2 \cdot d^2 + 2 \cdot d + 1$ for some positive integer d. From that, $d = \frac{-1+\sqrt{2 \cdot n-1}}{2}$. These two grids of vertices are transformed into two grids of edges (G and G^*) by substituting every vertex with two edges, one vertical and one horizontal. Quorums are defined as paths over the edge grids G and G^*. Thus, a read quorum can be built using a horizontal path in G and a write quorum using a horizontal path in G and a vertical path in G^*. A minimum path (horizontal

[3]When sites fail with probability p, the failure probability of quorum system F_p, $F_p \to 0$ when $p < \frac{1}{2}$ and $F_p \to 1$ when $p > \frac{1}{2}$.

Grid prot.	rq	wq	P_r	P_w^O	P_w^R
SQUARE	$\sqrt{n}$	$2 \cdot \sqrt{n} - 1$	$\frac{1}{\sqrt{n}}$	$\frac{1}{n}$	$\frac{2 \cdot \sqrt{n} - 2}{n}$
TRIANGLE	$\frac{-1+\sqrt{1+8 \cdot n}}{2}$	$\frac{-1+\sqrt{1+8 \cdot n}}{2}$	$\frac{-1+\sqrt{1+8 \cdot n}}{2 \cdot n}$	$\frac{1}{n}$	$\frac{-3+\sqrt{1+8 \cdot n}}{2n}$
PATHS	$\frac{1+\sqrt{2n-1}}{2}$	$\sqrt{2n-1}$	$\frac{1}{\sqrt{2n-1}-1}$	$\frac{1}{n}$	$\frac{\sqrt{2n-1}-1}{n}$

Fig. 9. Quorum sizes and probabilities of participating in a quorum of selected grid quorums.

or vertical) is of size $d + 1$. Thus, a read quorum is of size $d + 1 = \frac{1+\sqrt{2 \cdot n - 1}}{2}$ and a write quorum of size $2 \cdot d + 1 = \sqrt{2 \cdot n - 1}$. Since $d + 1 < \sqrt{n}$ for all values of d and n, this approach has better scalability than SQUARE. The advantage of PATHS lies in the extra flexibility added by using paths rather than columns. In SQUARE, we need a complete column. Hence, when all columns have a failed site, no quorum can be formed. In PATHS, the failures must occur all in the same row in the grid of size d or all in the same column in the grid of size $d+1$. Otherwise, the failed sites can be circumvented by building a path around them (at the cost of larger quorums).

Another variation on the rectangular grids are *hierarchical* grids [Kumar 1991]. A construction that is isomorphic to hierarchical grids has also been proposed by Naor and Wool [1998] in the form of the *AndOr* system. In a hierarchical grid, sites are organized in an h-level grid. At each level there is a grid. For instance, 16 sites can be configured into a 2-level grid of 2×2 at each level (Figure 8(D)). A read quorum is recursively formed by choosing an element of every column at each grid level. A write-only quorum is recursively formed by choosing at level i a full column of level $i - 1$ nodes. A write quorum is made combining any pair of read and write-only quorums. For rectangular grids, this results in a read quorum size of $\sqrt{n}$ and a write quorum size of $2 \cdot \sqrt{n} - 1$, that is, identical to its nonhierarchical counterpart. In terms of availability, a one-level rectangular grid has an asymptotic availability of 0. The hierarchical grid has an asymptotic availability of 1. However, this occurs only when n tends to infinity and, therefore, this difference in behavior is irrelevant in practice.

To study the characteristics of grid based quorums, we will focus our attention on SQUARE, TRIANGLE, and PATHS quorums systems. SQUARE represents the behavior of rectangular and hierarchical grids and are generally better than any other variations on this theme. TRIANGLE is the best representative of crumbling wall methods. PATHS are chosen as the best option among lattice based methods.

4.3 Scalability

Quorum sizes and probabilites are calculated in a similar way as for voting quorums. The results are shown in Figure 9. Applying these values to expressions (3) and (5) in Section 2.2, we can obtain the symmetric and asymmetric scale out factors for each one of the protocols, shown in Figure 10.

Grid prot.	so	$so(wo)$
SQUARE	$\frac{n}{\sqrt{n}\cdot(w+1)-w}$	$\frac{n}{w\cdot(1-2\cdot wo)\cdot(1-\sqrt{n})+\sqrt{n}}$
TRIANGLE	$\frac{2\cdot n}{-1+\sqrt{1+8\cdot n}}$	$\frac{2\cdot n}{3\cdot w\cdot(1-wo)-1+\sqrt{1+8\cdot n}\cdot(w\cdot wo+1-w)}$
PATHS	$\frac{2\cdot n}{(\sqrt{2\cdot n-1}-1)\cdot(w+1)+2}$	$\frac{2\cdot n}{(\sqrt{2\cdot n-1}-1)\cdot(w\cdot(2\cdot wo-1)+1)+2}$

Fig. 10. Scale out factors of selected grid quorums.

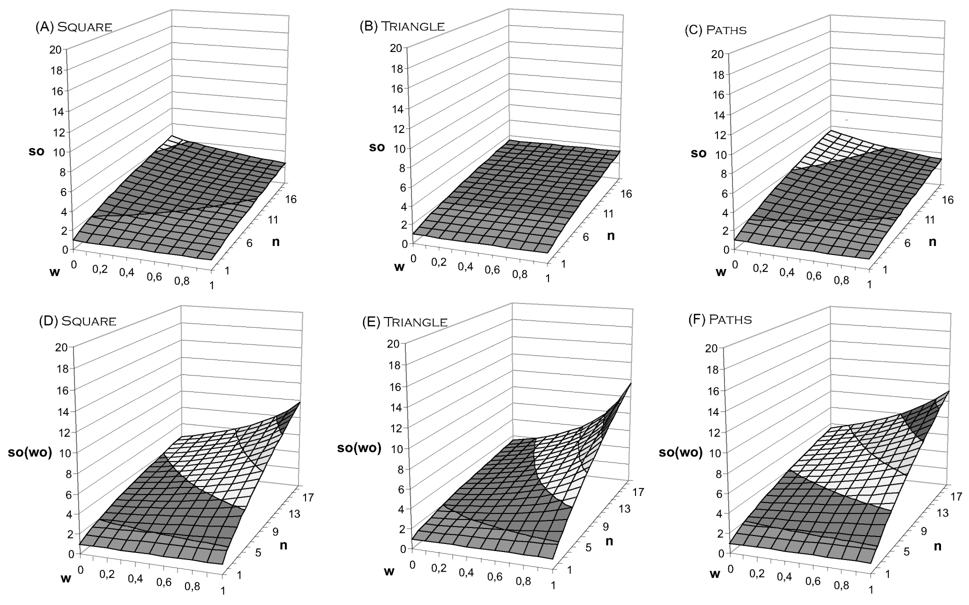

Fig. 11. Scalability (grid quorums, $wo = 0.15$): (A-C) symmetric, (D-F) asymmetric load.

Figure 11 shows the symmetric and asymmetric scale out factors as a function of the number of sites and the fraction of updates in the load (w).

For symmetric systems (Figures 11(A)-(C), the poor scalability of grid quorums is clear. Having the smallest quorum size, PATHS exhibits a slightly better behavior. TRIANGLE is not affected by w since quorum sizes are the same for read or writes. The slight advantage of PATHS and SQUARE over TRIANGLE for low values of w is due to the fact that read quorums in PATHS and SQUARE are smaller than write quorums.

The scalability of grid based quorums improves in asymmetric systems (Figures 11(D)-(F)). In an asymmetric system, scalability improves for write dominated loads since remote writes are less expensive. Due to its smaller quorum size, PATHS exhibits the best behavior for all values of n and w. Since SQUARE has smaller read quorums and larger write quorums than TRIANGLE, SQUARE performs better than TRIANGLE for low values of w and worse for high values of w.

Grid prot.	av_r	av_w
SQUARE	$(1-(1-p)^{\sqrt{n}})^{\sqrt{n}}$	$(1-(1-p)^{\sqrt{n}})^{\sqrt{n}} - (1-p^{\sqrt{n}} - (1-p)^{\sqrt{n}})^{\sqrt{n}}$
TRIANGLE	$1 - \sum_{i=1}^{\frac{-1+\sqrt{1+8\cdot n}}{2}} (1-p)^i \cdot \prod_{j=i+1}^{\frac{-1+\sqrt{1+8\cdot n}}{2}} (1-(1-p)^j - p^j)$	$1 - \sum_{i=1}^{\frac{-1+\sqrt{1+8\cdot n}}{2}} (1-p)^i \cdot \prod_{j=i+1}^{\frac{-1+\sqrt{1+8\cdot n}}{2}} (1-(1-p)^j - p^j)$
PATHS	$1-(1-p)^{d+1}$	$1-(1-p)^{d+1}-(1-p)^d$

	av
SQUARE	$(1-(1-p)^{\sqrt{n}})^{\sqrt{n}} - w \cdot (1-p^{\sqrt{n}} - (1-p)^{\sqrt{n}})^{\sqrt{n}}$
TRIANGLE	$1 - \sum_{i=1}^{\frac{-1+\sqrt{1+8\cdot n}}{2}} (1-p)^i \cdot \prod_{j=i+1}^{\frac{-1+\sqrt{1+8\cdot n}}{2}} (1-(1-p)^j - p^j)$
PATHS	$1-(1-p)^{d+1} - w \cdot (1-p)^d$

Fig. 12. Read and write quorum availabilities and overall availabilities of selected grid quorums.

4.4 Availability

The availability in grid quorums is usually determined by the probability that either no complete column or no complete path can be located. The latter implies that either an entire row (for vertical paths) or an entire column has failed (for horizontal paths). As an example, for rectangular grids, the write availability of a grid of size $r \times c$ is $(1-(1-p)^r)^c - (1-p^r-(1-p)^r)^c$ and the read availability is $(1-(1-p)^r)^c$ [Cheung et al. 1990]. The latter is the probability that every column has at least one working node. The former is the probability that every column has at least one working node and there is one column with all its nodes being up. The availability of SQUARE can be directly derived from there, with $sqrt(n)$ instead of r and c. The availability of PATHS is calculated in similar way but taking into account that one works with paths rather than columns or rows. We use the failure probability formula for crumbling walls in Peleg and Wool [1997] to calculate the availability of TRIANGLE, taking into account that row i has i elements (Figure 12).

Figure 13 shows the unavailability for the different grid quorums as a factor of w. Except for Figure 13(C), we only consider optimal configurations for each quorum.

The availability of grid quorums is dominated by a negative factor controlled by w. This is due to the larger size of write quorums which are roughly twice the size of read quorums in almost all grid quorums (except in TRIANGLE). This effect can be clearly seen in Figures 13(A) and 13(C) (for SQUARE) and Figure 13(D) (for PATHS). In rectangular grids, this effect can reach extreme proportions. Figure 13(C) shows the unavailability of two rectangular grids ($n = 14$ and $n = 22$). These two cases demonstrate the problems associated with the configuration of grids. The problem with systems of size $n = 14$ or $n = 22$ is that the only possible regular grids that can be constructed greatly distort the size of the quorums. When the load contains only read operations, the greater availability is due to the fact that read quorums are very small (2 nodes in both cases) and, therefore, the system becomes very resilient to failures. However, as soon as write quorums are needed, the availability of the system decreases dramatically.

Overall, the best availability is provided by TRIANGLE and PATHS. TRIANGLE has the advantage of having quorums of the same size for read and write operations.

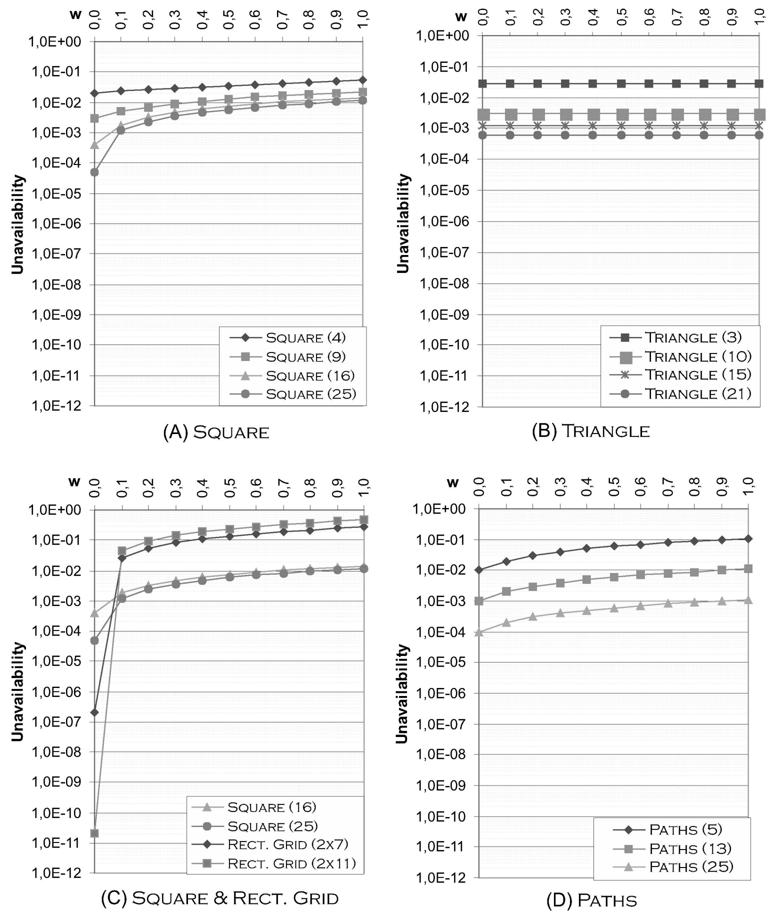

Fig. 13. Unavailability (grid quorums, $p = 0.9$): (A), (B), (D) optimal configurations; (C) including nonoptimal configurations. The numbers in parenthesis indicate the number of sites.

This makes it independent of the nature of the load. Unfortunately, the availability increases very slowly with system size. This is because, in TRIANGLE, if the last row fails, no quorum can be obtained. The relative size of this last row compared with the total number of nodes decreases as n increases, thereby limiting the advantage of having more sites. PATHS suffers from a more acute version of the same problem. The availability of PATHS increases monotonically with d ($n = 2 \cdot d^2 + 2 \cdot d + 1$). We gain one nine of availability for each unit increment of d. However, this gain implies a significant jump in system size. Thus, with respect to availability, TRIANGLE seems a better option as it reaches comparable availability levels with smaller system sizes.

4.5 Communication Overhead

Using expressions (7) and (8) from Section 2.4, the communication overhead in each of the grid quorums can be derived based on the different quorum sizes. The result is shown in Figure 14. Figure 15 compares the different protocols when

Grid prot.	Point-to-Point	Multicast
SQUARE	$2\cdot(\sqrt{n}-1)\cdot(\frac{3\cdot w}{o_w}+1-w)$	$\sqrt{n}\cdot(\frac{2\cdot w}{o_w}+1-w)$
TRIANGLE	$(-3+\sqrt{8\cdot n+1})\cdot(\frac{3\cdot w}{2\cdot o_w}+1-w)$	$\frac{-1+\sqrt{8\cdot n+1}}{2}\cdot(\frac{w}{o_w}+1-w)+\frac{w}{o_w}$
PATHS	$(-1+\sqrt{2\cdot n-1})\cdot(\frac{3\cdot w}{o_w}+1-w)$	$\frac{-1+\sqrt{2\cdot n-1}}{2}\cdot(\frac{2\cdot w}{o_w}+1-w)$

Fig. 14. Communication overhead for selected grid quorums.

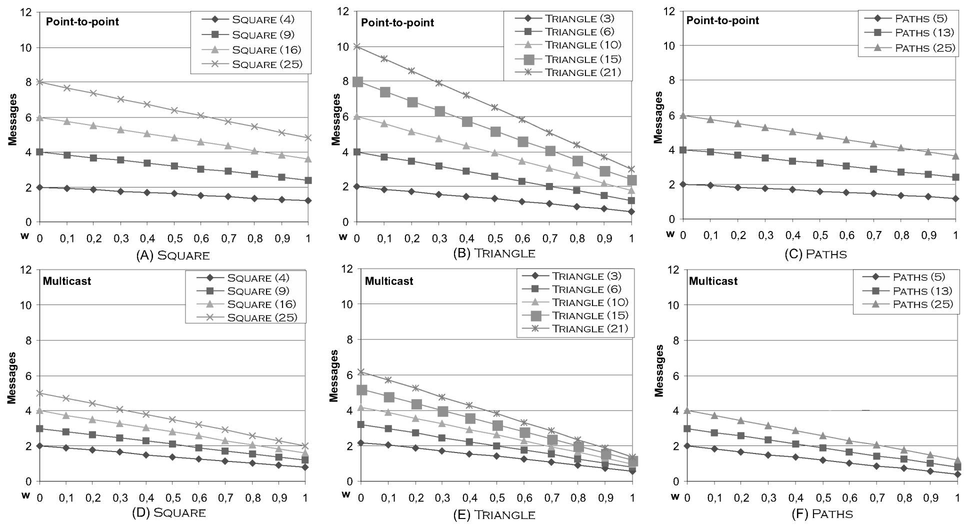

Fig. 15. Communication overhead in grid quorums: (A-C) point-to-point; (D-F) multicast. The numbers in parenthesis indicate the number of sites.

point-to-point communication is used (Figures 15(A)-(C)) and when multicast is used (Figures 15(D)-(F)). In terms of message overhead, PATHS does better than either SQUARE or TRIANGLE. This is due to the fact that the quorum size in PATHS is smaller than for the other two grid types.

4.6 Summary of Grid Quorums

From the comparison above, it is clear that PATHS is a better option than SQUARE and TRIANGLE. It has better scalability both in symmetric and asymmetric systems. It also exhibits better and more uniform availability. Finally, it has the lowest communication overhead for most load types.

Our analysis only looks at optimal configurations (squares, triangles, complete path structures). When nodes are removed or added, such configuration changes and both scalability and availability usually degrade. For instance, Naor and Wool [1998] provides mechanisms to find quorums for PATHS in any kind of configuration while still distributing the load equally among all nodes. However, scalability and availability might be considerably lower than that described above (Naor and Wool [1998] only provide lower bounds). Alternatively, in case of failure of an entire column, it would be feasible to transform the grid into an *SC-Grid* or *B-Grid* [Naor and Wool 1998], but quorum formation is different and the protocol would need to be changed, again changing the performance results.

A Square can be turned into a rectangular grid. For instance, in Square, if an entire column fails, one can conceivably remove that column and keep working with a rectangular grid. This is particularly useful when sites fail or parts of the system need to be taken off-line. This affects the behavior of the system (e.g., Figure 13(C)) but still results in a grid that can work under similar principles. Triangle can also be turned into crumbling wall grids keeping the same rules for forming quorums.

5. TREE QUORUMS

5.1 Basic Tree Quorum Construction

Tree quorums were first introduced in Agrawal and Abbadi [1990b] and further elaborated in Agrawal and Abbadi [1991, 1992]. Further tree protocols have been developed, mainly combining the concept of a tree structure with grid structures or majority voting. Similar to grid protocols, tree quorums impose a logical structure on the nodes in the system in order to reduce quorum sizes. The copies of an object are organized into a tree of height h and degree d, that is, each node has d children and the maximum length from the root to any leaf is h. Usually, analysis is only done for complete trees. In Agrawal and Abbadi [1992], the authors analyze incomplete d-trees (each node has at most d children). However, there exist many different incomplete d-trees for a given number of nodes, and both average quorum size and availability strongly depend on the structure of the tree. Hence, most of our calculations only hold for complete trees and only represent rough estimations for incomplete trees. A high degree of deviation can be expected in these cases.

A *tree quorum* $q = \langle l, b \rangle$ over a tree with height h and degree d is constructed as follows [Agrawal and Abbadi 1992]: The protocol tries to select the root of the tree and b children of the root. Furthermore, recursively for each selected child, b of its children have to be accessed, and so on, until a depth l is reached. In the case all nodes are accessible the quorum forms a tree of height l and degree b. If some node is inaccessible at depth h' from the root, the node is replaced by b tree quorums of height $l - h'$ starting from the children of the inaccessible node. In order to guarantee intersecting quorums, quorums must overlap both in height and degree. For a read quorum $rq = \langle l_r, b_r \rangle$ and write quorum $wq = \langle l_w, b_w \rangle$ to overlap we need $l_r + l_w > h$ and $b_r + b_w > d$, for two write quorums to overlap we have $2 \cdot l_w > h$ and $2 \cdot b_w > d$.

5.2 Alternative Tree Quorums

This general definition of tree quorums can now be refined to special instances that might favor one quorum type over the other or optimize availability. In Agrawal and Abbadi [1990b], the authors first introduced the ReadRoot protocol, in which a write quorum is $wq = \langle h, d/2 + 1 \rangle$ and a read quorum has dimensions $rq = \langle 1, (d+1)/2 \rangle$. Write operations are performed on a quorum

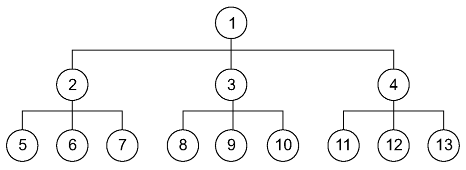

Fig. 16. Example of a tree structure: 13 copies organized into a tree of height 3 and degree 3.

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 the tree of Figure 16 with degree 3 and three levels, a write operation could access the set of copies {1, 2, 4, 6, 7, 11, 12}. 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 Figure 16 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. This means, ReadRoot has a majority approach regarding the degree parameter (both read and write operations must acquire a majority of d nodes), and a Rowaa approach regarding the height parameter (a read operation must only access one level in the best case, while write operations must access all levels).

In contrast, in MajTree, a majority approach is used both for the degree and the height parameters. That is, a read quorum is $rq = \langle (h+1)/2, (d+1)/2 \rangle$ and a write quorum is $wq = \langle h/2+1, d/2+1 \rangle$. This increases the availability of write operations (since write quorums can be built without the root) but also the access costs of read operations. Looking at Figure 16, possible read and write quorum are, for instance, {1, 2, 3} or, if the root is down, {2, 5, 6, 3, 8, 9}. However, if, for example, the root and both 2 and 3 are down, then neither read nor write quorums can be built.

Finally, in LogWrite read quorums have dimension $rq = \langle 1, d \rangle$ and write quorums are $wq = \langle h, 1 \rangle$. The smallest read quorum consists of only the root. If the root is down all children of the root must be accessed. If any of these nodes is not accessible, all its children must be accessed, etc. Example of read quorums in Figure 16 are {1}, {2, 3, 4} or {2, 3, 11, 12, 13}. Write operations must access one node in each level such that the nodes build a path from the root to one leaf (for instance {1, 4, 12} in Figure 16). This makes the approach similar to the grid protocol (the grid having a triangular structure) with the difference that no special measurements have to be taken to overlap write operations since all write quorums include the root.

Agrawal and Abbadi [1991] propose a binary tree quorum protocol for mutual exclusion that only considers write quorums. In general, a quorum is a path from the root to any of the leaves. However, in case of failure of the root or for load balancing purposes, not every quorum needs to access the root. As a result, the quorums are not s-uniform. Since it has similar characteristics as MajTree, we do not discuss it further.

Tree prot.	rq size	wq size	P_r	P_w^O	P_w^R
ReadRoot(3)	1	$2^{1+\lfloor log_3 n\rfloor}$	$\frac{1}{n}$	$\frac{1}{n}$	$\frac{2\cdot(2^{\lfloor log_3 n\rfloor}-1)}{n}$
ReadRoot(d)	1	$\frac{(d+1)\cdot n^{\lfloor log_3 \frac{d+1}{2}\rfloor}-2}{d-1}$	$\frac{1}{n}$	$\frac{1}{n}$	$\frac{d+1}{d-1}\cdot\frac{n^{\lfloor log_d \frac{d+1}{2}\rfloor}-1}{n}$
MajTree(3)	$2^{\frac{\lfloor log_3 n\rfloor+3}{2}}-1$	$2^{\frac{\lfloor log_3 n\rfloor+3}{2}}-1$	$\frac{2^{\frac{\lfloor log_3 n\rfloor+3}{2}}-1}{n}$	$\frac{1}{n}$	$\frac{2^{\frac{\lfloor log_3 n\rfloor+3}{2}}-2}{n}$
MajTree(d)	$\frac{\frac{d+1}{2}^{\frac{\lfloor log_d n\rfloor+3}{2}}-2}{d-1}$	$\frac{\frac{d+1}{2}^{\frac{\lfloor log_d n\rfloor+3}{2}}-2}{d-1}$	$\frac{\frac{d+1}{2}^{\frac{\lfloor log_d n\rfloor+3}{2}}-2}{n\cdot(d-1)}$	$\frac{1}{n}$	$\frac{\frac{d+1}{2}^{\frac{\lfloor log_d n\rfloor+3}{2}}-(d+1)}{n\cdot(d-1)}$
LogWrite	1	$\lfloor log_d n\rfloor+1$	$\frac{1}{n}$	$\frac{1}{n}$	$\frac{\lfloor log_d n\rfloor}{n}$

Fig. 17. Quorum sizes and participation probabilities for selected tree protocols.

5.3 Scalability

The tree quorum systems are a special case in that sites generally do not have the same degree. All other protocols (except Triangle) ensure that all sites do about the same amount of work without any special arrangement. For tree quorums, the root and upper level nodes have a much higher load than the leaf nodes, and there are no obvious mechanisms to automatically distribute the load. Only MajTree offers the possibility to alleviate the load of the upper level nodes at the price of generating bigger quorum sizes. ReadRoot and LogWrite require each write quorum to access the rool. 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 and resolve the expression for ternary trees. For analysis purposes, we only consider cases in which d is odd and larger than 1 (i.e., $d = 2\cdot m - 1, m > 1$). If the number of nodes is given by n, we can approximate the height h as $h = \lfloor log_d n\rfloor + 1$ and $n = \sum_{i=0}^{\lfloor log_d n\rfloor} d^i$ (assuming the root is at level 0, its children at level 1, ...).

Quorum sizes are given by the width and length of the quorum subtree. If a quorum is determined by $q = \langle l, b\rangle$, then the size of the quorum can be calculated by: $\sum_{i=0}^{l} b^i$. Figure 17 summarizes the quorum sizes for the ReadRoot, MajTree, and LogWrite protocols. The table also provides the resulting probabilities of being in a read (P_r) quorum, being the originator of a write (P_w^O), and being a remote node in a write operation (P_w^R). The scalability expressions (3) and (5) in Section 2.2 for each of the protocols are depicted in Figure 18.

Tree prot.	so	so(wo)
ReadRoot(3)	$\frac{n}{1+2\cdot w\cdot(2^{\lfloor log_3 n\rfloor}-1)}$	$\frac{n}{1+w\cdot wo\cdot 2\cdot(2^{\lfloor log_3 n\rfloor}-1)}$
ReadRoot(d)	$\frac{n}{1+w\cdot\frac{d+1}{d-1}\cdot(n^{\lfloor log_d \frac{d+1}{2}\rfloor}-1)}$	$\frac{n}{1+w\cdot wo\cdot\frac{d+1}{d-1}\cdot(n^{\lfloor log_d \frac{d+1}{2}\rfloor}-1)}$
MajTree(3)	$\frac{n}{2^{\frac{\lfloor log_3 n\rfloor+3}{2}}-1}$	$\frac{n}{w+w\cdot wo\cdot(2^{\frac{\lfloor log_3 n\rfloor+3}{2}}-2)+(1-w)(2^{\frac{\lfloor log_3 n\rfloor+3}{2}}-1)}$
MajTree(d)	$\frac{n\cdot(d-1)}{\frac{d+1}{2}^{\frac{\lfloor log_d n\rfloor+3}{2}}-2}$	$\frac{n}{w+w\cdot wo\cdot\frac{\frac{d+1}{2}^{\frac{\lfloor log_d n\rfloor+3}{2}}-(d+1)}{d-1}+(1-w)\cdot\frac{\frac{d+1}{2}^{\frac{\lfloor log_d n\rfloor+3}{2}}-2}{d-1}}$
LogWrite	$\frac{n}{1+w\cdot\lfloor log_d n\rfloor}$	$\frac{n}{1+w\cdot wo\cdot\lfloor log_d n\rfloor}$

Fig. 18. Scale out factors for selected tree protocols.

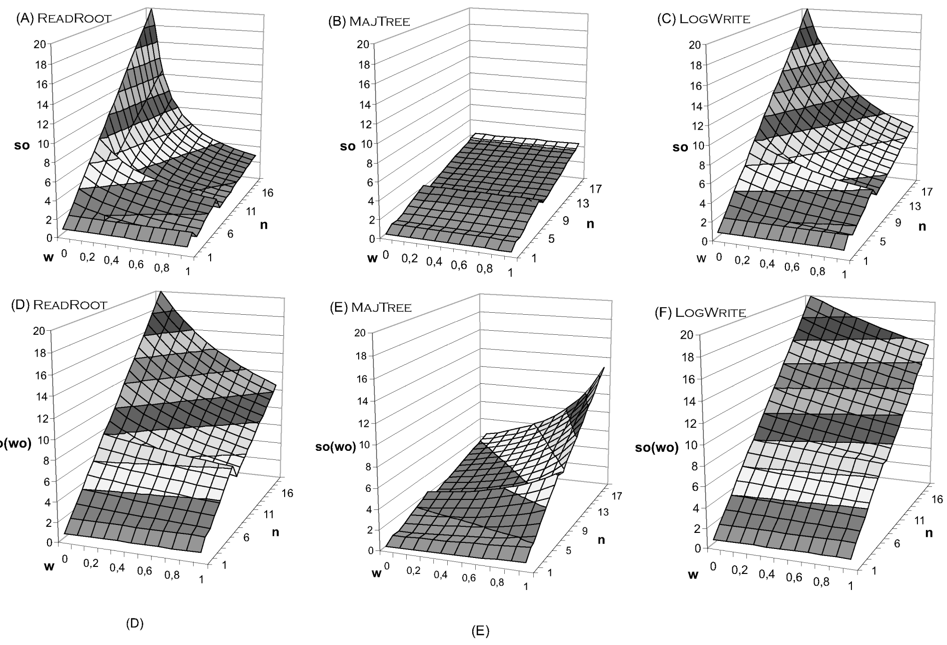

Fig. 19. Scalability of tree protocols: (A-C) symmetric load; (E-F) asymmetric load ($wo = 0.15$).

Figure 19 depicts the scalability for trees with degree $d = 3$ for both the symmetric (A)-(C) and asymmetric case (D)-(E). In the symmetric case, MajTree has the worst scalability of all three protocols loads due to the high costs of read operations. It only outperforms ReadRoot for very high update rates. ReadRoot has very good scalability for high read rates but degrades very fast for write intensive applicaitons. LogWrite has the best scalability. It even scales reasonably well for write intensive applications. The reason is that reads are local and write quorums are logarithmic on the number of nodes in the system. In the asymmetric case, the results are generally better, especially for high update rates. The relative performance between the protocols is similar to the symmetric case. We would like to note, however, that this performance can only be achieved if we are able to partition our data in equally active regions and build individual trees for each of the regions such that each node in the system experience the same load.

5.4 Availability

The availabilities of read and write operations can be expressed in form of recurrence relations over the height of the tree. Let $av(h)\langle l, b\rangle$ be the availability of an operation that requires a tree quorum of length l and width b in a tree of height h and degree d (we again assume that d is odd and $d > 1$, that is, $\frac{d+1}{2}$ builds a majority of children). The availability in a tree of height $h+1$ can be expressed by Agrawal and Abbadi [1992]:

$$\begin{aligned} av(h+1)\langle l, b\rangle = {} & \mathit{Probability}(\mathit{root\ is\ up}) \\ & \cdot [\mathit{Availability\ of\ b\ subtrees\ with\ } a(h)\langle l-1, b\rangle] \\ & + \mathit{Probability}(\mathit{root\ is\ down}) \\ & \cdot [\mathit{Availability\ of\ b\ subtrees\ with\ } a(h)\langle l, b\rangle] \end{aligned}$$

Taking p as the probability that a node is available, we obtain:

$$\begin{aligned} av(h+1)\langle l, b\rangle = {} & p \cdot \sum_{i=0}^{d-b} \binom{d}{b+i} \cdot (av(h)\langle l-1, b\rangle)^{b+i} \cdot (1 - av(h)\langle l-1, b\rangle)^{d-b-i} \\ & + (1-p) \cdot \sum_{i=0}^{d-b} \binom{d}{b+i} (av(h)\langle l, b\rangle)^{b+i} \cdot (1 - av(h)\langle l, b\rangle)^{d-b-i} \end{aligned}$$

For MAJTREE we have to replace b by $\frac{d+1}{2}$ and l by $\frac{(h+1)+1}{2}$ for read quorums and $\frac{(h+1)}{2} + 1$ for write quorums. For READROOT and LOGWRITE the general equations can be simplified and we can express the following availabilities for read (av_r) and write (av_w) operations:

$$\begin{aligned} av_r(\mathit{ReadRoot}) &= \mathit{Prob}(\mathit{root\ is\ up}) + \mathit{Prob}(\mathit{root\ is\ down}) \\ &\quad \cdot \mathit{Prob}(\mathit{a\ majority\ of\ subtrees\ is\ read\ available}) \\ av_w(\mathit{ReadRoot}) &= \mathit{Prob}(\mathit{root\ is\ up}) \cdot \mathit{Prob}(\mathit{a\ majority\ of\ subtrees\ is\ write\ available}) \\ av_r(\mathit{LogWrite}) &= \mathit{Prob}(\mathit{root\ is\ up}) + \mathit{Prob}(\mathit{root\ is\ down}) \\ &\quad \cdot \mathit{Prob}(\mathit{all\ subtrees\ are\ read\ available}) \\ av_w(\mathit{LogWrite}) &= \mathit{Prob}(\mathit{root\ is\ up}) \cdot \mathit{Prob}(\mathit{at\ least\ one\ subtree\ is\ write\ available}) \end{aligned}$$

Given that the availability of a tree of height $h = 1$ (one node) is $av(1) = p$ for all tree protocols and quorum types, Figure 20 indicates the recurrence equations for all three tree protocols for $h > 1$.

Figure 21 shows the unavailability of the different tree protocols as a factor of w. We only consider complete tree configurations. As with the grid protocols, w influences the availability of the system negatively. This holds for all tree protocols (except for MAJTREE of height $h = 3$ where read and write quorums have the same size). For READROOT, the availability of read operations increases with the number of nodes while the availability of write operation decreases. For MAJTREE the behavior depends on whether the height is even or odd. In LOGWRITE increasing the number of nodes helps for read operations, write operations are

Tree prot.	$av_r(h+1)$
ReadRoot	$p + (1-p) \cdot \sum_{i=0}^{\frac{d-1}{2}} \binom{d}{\frac{d+1}{2}+i} \cdot av_r(h)^{\frac{d+1}{2}+i} \cdot (1 - av_r(h))^{\frac{d-1}{2}-i}$
MajTree	$p \cdot \sum_{i=0}^{\frac{d-1}{2}} \binom{d}{\frac{d+1}{2}+i} \cdot (av_r(h)\langle \frac{h+1}{2}, \frac{d+1}{2} \rangle)^{\frac{d+1}{2}+i} \cdot (1 - av_r(h)\langle \frac{h+1}{2}, \frac{d+1}{2} \rangle)^{\frac{d-1}{2}-i}$ $+(1-p) \cdot \sum_{i=0}^{\frac{d-1}{2}} \binom{d}{\frac{d+1}{2}+i} \cdot (av_r(h)\langle \frac{(h+1)+1}{2}, \frac{d+1}{2} \rangle)^{\frac{d+1}{2}+i} \cdot (1 - av_r(h)\langle \frac{(h+1)+1}{2}, \frac{d+1}{2} \rangle)^{\frac{d-1}{2}-i}$
LogWrite	$p + (1-p) \cdot av_r(h)^d$

	$av_w(h+1)$
ReadRoot	$p \cdot \sum_{i=0}^{\frac{d-1}{2}} \binom{d}{\frac{d+1}{2}+i} \cdot av_w(h)^{\frac{d+1}{2}+i} \cdot (1 - av_w(h))^{\frac{d-1}{2}-i}$
MajTree	$p \cdot \sum_{i=0}^{\frac{d-1}{2}} \binom{d}{\frac{d+1}{2}+i} \cdot (av_w(h)\langle \frac{h}{2}+1, \frac{d+1}{2} \rangle)^{\frac{d+1}{2}+i} \cdot (1 - av_w(h)\langle \frac{h}{2}+1, \frac{d+1}{2} \rangle)^{\frac{d-1}{2}-i}$ $+(1-p) \cdot \sum_{i=0}^{\frac{d-1}{2}} \binom{d}{\frac{d+1}{2}+i} \cdot (av_w(h)\langle \frac{h+1}{2}+1, \frac{d+1}{2} \rangle)^{\frac{d+1}{2}+i} \cdot (1 - av_w(h)\langle \frac{h+1}{2}+1, \frac{d+1}{2} \rangle)^{\frac{d-1}{2}-i}$
LogWrite	$p \cdot \sum_{i=0}^{d-1} \binom{d}{1+i} \cdot av_w(h)^{1+i} \cdot (1 - av_w(h))^{d-1-i}$

Fig. 20. Availability for selected tree protocols.

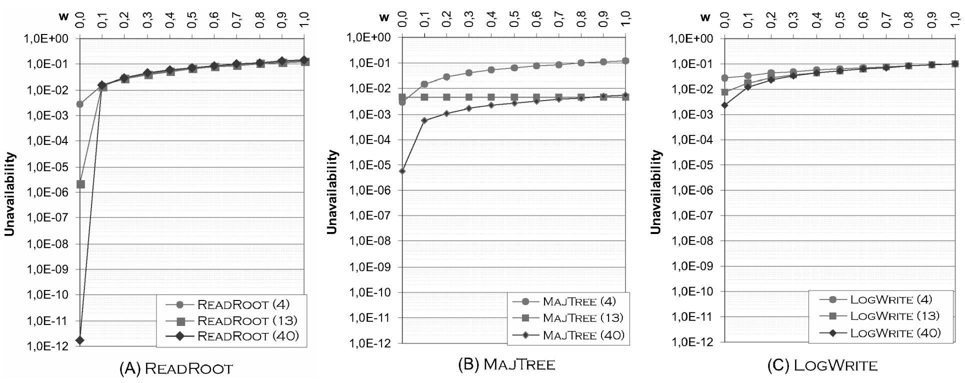

Fig. 21. Unavailability of tree protocols ($p = 0.9$). The numbers in parenthesis indicate the number of sites.

always close to the availability of the root, since on each further level, only one out of d nodes needs to be available.

5.5 Communication Overhead

The communication overhead for the tree quorums according to Definitions 7 and 8 given in Section 2.4 for both multicast and point-to-point is given in Figure 22. For ReadRoot and LogWrite, communication overhead increases with the update rate whereby the overhead of LogWrite stays considerable small due to the small quorum sizes. The performance improvements of multicast over point-to-point are obvious in these cases. The behavior of MajTree depends strongly on the configuration. At height 2, a read quorum consists of only one node favoring read intensive applications. At height 3, read and write quorums have the same size and therefore, increasing update rates decreases

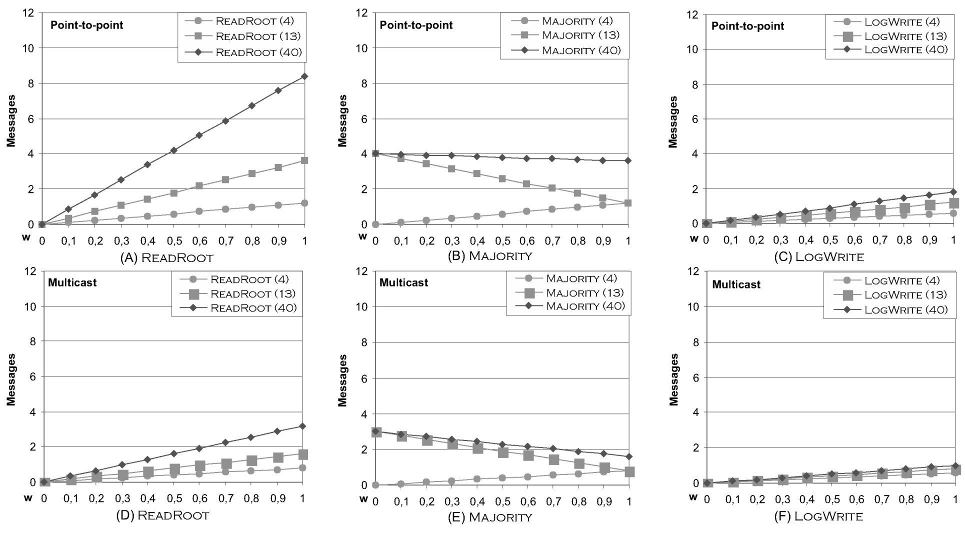

Fig. 22. Communication overhead of tree protocols for $o_w = 5$: (A-C) point-to-point; (D-F) multicast. The numbers in parenthesis indicate the number of sites.

the message overhead due to the capacity to bundle messages. At height 4, write quorums are bigger than read quorums. Still, the high costs for remote read operations lead to higher message costs at high read rates.

5.6 Summary of Tree Protocols

Within the tree protocols, LogWrite provides the overall best performance. Although MajTree has better availability, the low scalability and high message rates make it a very unattractive candidate.

However, there are a couple of issues that must be taken into consideration when applying tree structures. In very few cases the number of nodes in the system leads to a complete tree. Although incomplete trees can be used in exactly the same way as complete trees [Agrawal and Abbadi 1992], their scalability and availability varies, not only between trees with different number of nodes but also within different tree configurations of the same size. Some work has been done on how to balance the load in trees [Wool 1996; Naor and Wool 1998] that can help to overcome this shortcoming.

As mentioned earlier, the indicated good scalability for ReadRoot and LogWrite can only be achieved when it is possible to build different trees for different data partitions, and to be able to distribute the load equally across all sites. We doubt that this can be achieved in practice.

Additionally, tree protocols are not suitable in dynamic environments. For instance, when one node fails permanently it must be replaced by another node, otherwise one partition is not going to be available anymore (the node was root of one partition) and all other trees in which the node was member change their structure and with it, their availability and scalability. Similarly constraints hold when new nodes join the system.

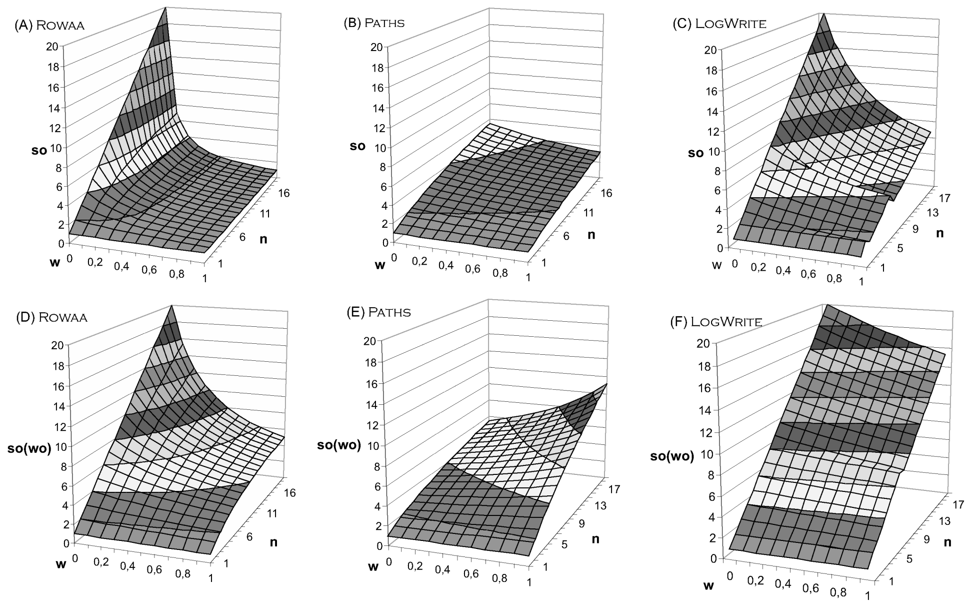

Fig. 23. Scalability of Rowaa, Paths and LogWrite.

6. OVERALL COMPARISON

In what follows, we compare the best solutions found in each category (Rowaa, Paths, and LogWrite).

6.1 Scalability

Figure 23 compares the scalability for these three protocols. LogWrite by far outperforms the other two protocols for the entire range. Reading local data and having only a logarithmic size of write quorums is the particular strength of LogWrite. Paths is by far the worst protocol and only performs better than Rowaa for high update rates in asymmetric environments.

Unfortunately, LogWrite can only scale well if the load is distributed evenly across all nodes. Since the protocol is not fair per se, all write quorums must access the root, a smart a priori data partitioning is necessary to be able to take advantage of this scalability. This means, the graph depicts only the best case and the real scalability will be smaller. In the worst case, scale-out will be one for all workloads. Finding a smart data distribution is, however, not easy, and we are not aware of any algorithm that would tackle this problem.

In Rowaa and Paths, it is quite simple to achieve the scalability depicted in the figure. Additionally, both Rowaa and Paths are fair. Hence, implementing load balancing is trivial (e.g., send the next transaction to the node with the lowest current CPU load).

6.2 Availability

Figure 24 depicts the availability of the three candidates. In this case Rowaa is by far the best protocol, showing the fastest improvement when the number of

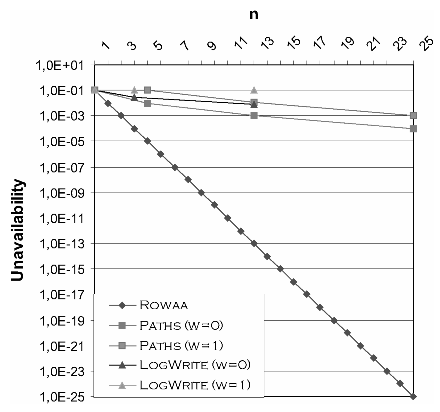

Fig. 24. Unavailability of Rowaa, Paths and LogWrite.

nodes in the system increases. It also only depends on the number of nodes and not on the type of application (read or write intensive).

Paths has acceptable availability, however, a considerable number of nodes must be added to the system to decrease unavailability by one 9, and there is always one 9 difference between the unavailability of read and write operations. LogWrite has the worst availability and does not provide fault-tolerance for write operations: if the root fails, then write operations cannot be performed. If we partition the data, and each node is root for a different partition, then only the write operations on the data for which the failed node is the root, cannot be accessed anymore. This has two implications: First, even if one root fails, we can still write the data for which the node was not the root. Second, there exist many roots and if any of the roots fail, some data partition cannot be written anymore. Depending on the application this will lead to higher or smaller availability as depicted in the figure. If the application is still functional even if some data is no more writable, availability is higher. If the application requires all data to be accessible for update at any time, then the availability is much smaller because all roots must be up for the system to be operational. The more roots there are, the less likely this will happen.

6.3 Communication Overhead

Figure 25 compares the communication overhead of the three protocols. The communication behavior is similar to the scalability results.

LogWrite has the all-over best performance scaling very well over the entire range. For instance, while Rowaa's performance for large systems deteriorates at high update rates, LogWrite keeps the communication overhead very small, independently of whether multicast is available or point-to-point must be used. Paths has the opposite behavior than the other two protocols. While Rowaa and LogWrite have higher overhead for updates, Paths has higher overhead for read operations. However, LogWrite outperforms Paths even for update rates of 100%. If the choice is between Rowaa and Paths, then we have to look at the

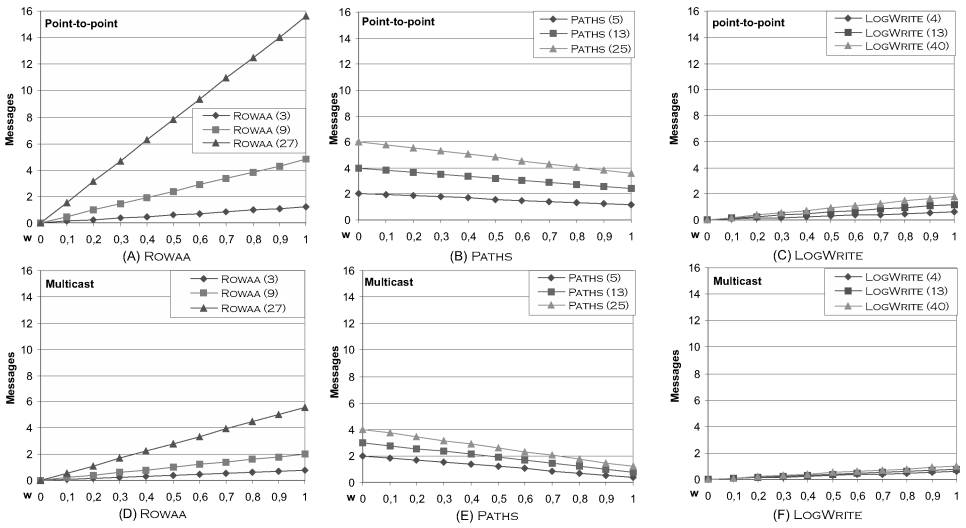

Fig. 25. Communication overhead of Rowaa, Paths and LogWrite. The numbers in parenthesis indicate the number of sites.

application to decide which protocol is better. Since most applications will have a tendency to be read intensive, Rowaa will be the better choice.

6.4 Configuration Density

Configuration density describes the behavior of the system when nodes are added or removed. Other changes include the addition of new data to the system or the change in access patterns in the workload. Thus, Rowaa, smoothly adapts to any change. If nodes are added, the availability increases by a factor easy to calculate and the scalability and communication equations are correct for all system sizes. New data can simply be added by performing "Insert" statements at all sites (just another write), and since all nodes have the same access right to all data, changes in the access patterns can be handled by simple load distribution algorithms.

For Paths, the behavior is less smooth. If the system size n is not of the form $n = 2 \cdot d^2 + 2 \cdot d + 1$, we have to build a Paths structure in which some nodes appear twice, or where quorum sizes in general become larger. As a result, some points in the grid become more critical and more loaded than others. This means that in the case of incomplete grids, we have to do a smarter load distribution. Naor and Wool [1998] propose algorithms for such load redistribution. However, they are cost intensive and might not lead to optimal scalability.

The same holds for LogWrite. Adding a new node changes the form of the tree. Although incomplete trees do not necessarily have less scalability or availability, it is a cumbersome calculation to change a tree with n nodes into a tree with $n+1$ such that the tree has optimal availability and smallest quorum size of all trees of size $n+1$. Additionally, whenever we add a new node we have to create a new tree with this new node as the root and repartition our data from n trees to $n+1$ trees in order to achieve optimal load distribution. For the same reason, if we new data is added to the system or the access patterns change, the data partitions must be recomputed.

7. RELATED WORK

There are many important results on the availability of quorum systems and most of the papers that study quorum systems compare the availability of different quorum protocols.

For systems with homogeneous site availability, Barbará and Garcia-Molina [1987] show that for $p > 0.5$ the most available quorum system is MAJORITY. For $p < 0.5$, Peleg and Wool [1995] show that the singleton (monarchy) exhibits the best availability. In that paper the authors study the availability of several quorums systems (MAJORITY, wheel, TRIANGLE, LOGWRITE, SQUARE, ...). The authors provide the number of quorums, the minimal quorum cardinality and the failure probability. The availability figures presented in our paper show what was predicted by these theoretical results. A detailed comparative study of the availability of exclusive quorums (equivalent to write quorums) can be found in Wool [1996]. In this comparison, the availability formulas for exclusive quorums are derived and those quorums exhibiting a Condorcet failure probability are identified.

For systems with heterogeneous site availability, if all sites have an availability $p > 0.5$, Tong and Kain [1988] and [Spasojevic and Berman 1994] show that the optimal quorum system is weighted voting and provide a formula to obtain the optimal weights. Finally, for the general case of systems with heterogeneous site availability where $0 < p < 1$, Amir and Wool [1998] prove that, if there are sites with $p > 0.5$, then all sites with $p < 0.5$ should be set zero weights (dummy copies) except when all sites $p < 0.5$, in which case the monarchy is the optimal quorum with one of the sites with the highest availability as king.

Amir and Wool [1996] present an empirical study of quorum systems over the Internet. The study was conducted with a 14-site system distributed among two different geographic sites and three Internet segments during 6 months. Fourteen dynamic and static quorums systems were evaluated. The study uses three quality measures: system unavailability (there is no available quorum), unaccessibility (the probability that the network component where a site resides does not contain a quorum) and worst unaccessibility. The authors show that dynamic quorum systems behave better than static ones as predicted by analytical and simulation studies. An important contribution of this paper is that it proves empirically that the traditional assumptions in availability studies, independent failures and absence of partitions, are invalid. More particularly, their experiments show that the independent failure assumption does not hold in LANs, and the absence of partitions does not hold in WANs.

Wool [1998] argues that quorums might become a viable option for replicated database systems in the near future. A first reason is that due to CPU and network improvements, remote reads do not lead to much higher response times than local reads, especially in LAN's. Secondly, by analyzing the load in a similar way as our symmetric load analysis in Section 2.2, the author concludes that quorums are better than ROWAA at already moderate update rates (e.g., 25% in Maekawa's quorum system [Maekawa 1985] of 13 sites). Our results confirm these numbers for symmetric systems. However, our asymmetric analysis shows that asymmetric writes lower the impact of large write quorums even for high update rates over 50%, and hence, favor the ROWAA approach for a very wide range of applications.

Rangarajan et al. [1995] propose a quorum algorithm (subgroups) whose resiliency (availability) is compared with MAJORITY and Maekawa's algorithm [Maekawa 1985]. The authors also compare the quorum sizes (best and worse case) and the worst case message overhead (when failures occur) for the tree algorithm [Agrawal and Abbadi 1989], HQC and grid-set [Agrawal and Abbadi 1990a]. They assume point-to-point communication, and contact a quorum for each operation (no deferred updates).

The capacity of some quorum systems (MAJORITY, Maekawa's algorithm, HQC, and subgroups [Rangarajan et al. 1995]) is studied by Rangarajan et al. [1993]. The capacity is defined as the number of operations a system can perform on average. The capacity takes into account the availability and the time needed to perform an update operation and to process the messages needed to perform that operation. The authors show that for MAJORITY, the capacity cannot be increased by a factor greater than $\frac{1}{p}$ (being p the probability of a node being active). For Maekawa's algorithm, it is shown that the capacity increases up to a certain point and then it decreases.

According to Naor and Wool [1998], most of the previous analysis do not distinguish between the properties of the quorum system and the properties of the strategy that chooses which quorum to access. Naor and Wool define a strategy as the frequency of picking each quorum. A strategy induces the frequency of accessing each element, which is the load on that element. The load on a given quorum system is the minimal load on the busiest element induced by the best possible strategy. The authors calculate the minimum load for voting systems and some tree protocols. They also present four algorithms that have optimal or near optimal load and high availability (PATHS, B-Grid, SC-grid and AndOr). The paper studies the protocols for mutual exclusion (write operations). The load on an element is what we have called the probability of a site to participate in a (read) write quorum. For those quorum systems that are not (s, d)-fair the scalability provided must be considered an upper bound. Wool [1996] provides a more exhaustive comparison of the load of mutual exclusion quorums.

Keum et al. [1995] carry out a simulation of the performance for ROWA, MAJORITY, primary copy and the READROOT protocols. The performance measures are the response time for both read and write transactions and the ratio of the total number of committed transactions to the total number of transactions. Their simulation assumes that locks must be acquired at each site in a quorum before performing a read or write operation. They show that if the read transaction ratio is low, primary copy and READROOT protocols outperform the other protocols. MAJORITY has poor performance under all simulation conditions. The performance of ROWA varies with system load and read transaction ratio.

Liu et al. [1997] compare the performance of primary copy and MAJORITY protocols with no replication. The simulation measures data availability, response time and throughput with and without failures for three- and five-fold replication. In the primary copy, protocol updates are deferred until the 2PC is executed.

Nicola and Jarke [2000] present an analytical model for symmetric replicated database systems. The main innovation is the study of replication in the context of two dimensions with integrated communication. These dimensions are (1) the percentage of replicated data, and (2) the degree of replication of the data. The first dimension varies from zero replicated items to the replication of all the items, whilst the second dimension varies from two-fold partial replication to full replication. The analytical model is based on a primary copy approach, and analyzes throughput and response time. Their results depicts configurations in which either the network of the database become the bottleneck.

Message overhead is studied in Saha et al. [1996]. The authors study the average case message overhead for MAJORITY, rectangular grids, LOGWRITE, HQC and RST [Rangarajan et al. 1995] protocols. The average message overhead for point-to-point communication is calculated taken into account the presence of failures. They also consider the trade-off between message overhead and availability.

Kumar and Segev [1993] study how to assign votes to minimize the communication cost taken into account the availability of the quorum protocols. The model assumes that locks must be acquired at each site in a quorum before performing a read or write operation. They do not consider the availability of multicast primitives. They study the communication cost for equal and different vote assignments, with and without tolerating failures, and for different availability thresholds.

Kumar [1990] compares the communication cost and availability of HQC and MAJORITY. The communication cost is studied when link failures occur for point to point communication.

Ingols and Keidar [2001] perform a simulation to evaluate the effect of the interruptions on the availability of dynamic voting algorithms. Dynamic voting consists in selecting a new primary component adaptively. Interruptions take into consideration the fact that the process to form a new primary component does not always succeed.

Many efforts have been devoted to compare different performance criteria (throughput, response time, message cost, abort rate, and deadlocks) for distributed and replicated databases [Ciciani et al. 1990; Carey and Livny 1986; Gray and Reuter 1993; Anderson et al. 1998; Hwang et al. 1996]. However, none of those efforts take into account quorum protocols.

In terms of the scalability of replication protocols, we are not aware of any approach analyzing asymmetric systems, i.e., all citations above refer to analysis of symmetric systems. Furthermore, we have not found any discussion regarding the complexity and technical problems of performing database read operations (SQL `select` statements) within a quorum compared to simply executing them at a single site. Finally, to the best of our knowledge, none of the papers that takes into account the communication overhead uses multicast primitives and only [Liu et al. 1997] consider deferred update propagation for the primary copy protocol.

8. CONCLUSIONS

In this article, we have compared quorum based data replication protocols among themselves and with the conventional read one-write all available approach. Although quorum protocols are often proposed as the means to improve the performance and availability of replicated databases, the results presented in this paper raise important questions regarding their applicability in practice.

Scalability. ROWAA and LOGWRITE are, in principle, the best choices for read intensive environments in both symmetric and asymmetric systems. For very write intensive environments (close to write only applications), PATHS outperforms ROWAA. However, 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 70/30 or even 80/20 ratio in most cases). Thus, most applications greatly benefit from local reads. This makes ROWAA and LOGWRITE better choices for systems that must support a wide range of loads. To decide between these two protocols, one needs to look at load balancing. In ROWAA, the load is automatically balanced by simply directing transactions uniformly to all nodes. In LOGWRITE, it is theoretically possible to balance the load using data partitions and assuming disjoint access patterns. In practice, unfortunately, this is not a very realistic approach and LOGWRITE creates serious load balancing problems that render it infeasible in many scenarios. Therefore, ROWAA is realistically speaking the best option from a scalability point of view.

Availability. ROWAA provides the best possible availability: the availability increases linearly, in a logarithmic scale, with each additional replica added to the system. Moreover, the availability does not depend on the type of workload (read intensive or write intensive). The simple version of ROWAA does not tolerate network partitions but it can be extended so that it can cope with them. In that case, the availability of ROWAA is the same as that of MAJORITY. Since MAJORITY is the second best protocol in terms of availability, we conclude that ROWAA is the best option 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, LOGWRITE has the best behavior in terms of message overhead. ROWAA is expensive in terms of messages for write intensive applications, however, never to the point of rendering it impractical.

Configuration. ROWAA does not impose any restriction in the number of nodes. It also requires very little in terms of configuring the system. Changes in the configuration can take place dynamically without having to change the protocol. In this regard, ROWAA is a much better option than any of the other protocols discussed.

The obvious conclusion from these results is that ROWAA is the best choice for a wide range of application scenarios. It offers good scalability (within the limitations of replication protocols), very good availability, and an acceptable communication overhead. It also has the significant advantage of being very simple to implement and very easy to adapt to configuration changes. For very peculiar loads and configurations, it is possible that some variation of quorum does better than ROWAA. The analyses provided in this article clearly identify these situations and can serve as a guide to system designers for the few cases in which ROWAA is not adequate.

REFERENCES

AGRAWAL, D. AND ABBADI, A. E. 1989. An efficient solution to the mutual exclusion problem. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. (Edmonton, Alb., Canada). ACM, New York, 193–200.

AGRAWAL, D. AND ABBADI, A. E. 1990a. Exploiting logical structures in replicated databases. *Information Processing Letters 33,* 5, 250–260.

AGRAWAL, D. AND ABBADI, A. E. 1990b. The tree quorum protocol: An eficient approach for managing replicated data. In *Proceedings of the International Conference on Very Large Databases (VLDB)* (Brisbane, Australia). Morgan-Kaufmann, San Mateo, Calif., 243–254.

AGRAWAL, D. AND ABBADI, A. E. 1991. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Trans. Comput. Syst. 9*, 1 (Feb.), 1–20.

AGRAWAL, D. AND ABBADI, A. E. 1992. The generalized tree quorum protocol: An efficient approach for managing replicated data. *ACM Trans. Datab. Syst. 17*, 4 (Dec.), 689–717.

AHAMAD, M. AND AMMAR, M. H. 1989. Performance characterization of quorum-consensus algorithms for replicated data. *IEEE Trans. Softw. Eng. 15*, 4 (Apr.), 492–496.

AMIR, Y. AND WOOL, A. 1996. Evaluating quorum systems over the internet. In *Proceedings of the IEEE International Conference on Fault-Tolerant Computing Systems (FTCS)* (Sendai, Japan). IEEE Computer Society Press, Los Alamitos, Calif., 26–35.

AMIR, Y. AND WOOL, A. 1998. Optimal availability quorums systems: Theory and practice. *Inf. Proc. Lett. 65*, 5 (Mar.), 223–228.

ANDERSON, T. A., BREITBART, Y., KORTH, H. F., AND WOOL, A. 1998. Replication, consistency, and practicality: Are these mutually exclusive? In *Proceedings of the ACM SIGMOD Conference* (Seattle, Wash.). ACM, New York, 484–495.

BACON, J. 1997. *Concurrency Systems*. Addison-Wesley, Reading, Mass.

BARBARÁ, D. AND GARCIA-MOLINA, H. 1987. The reliability of vote mechanisms. *IEEE Trans. Comput. 36*, 10 (Oct.), 1197–1208.

BARBARÁ, D., GARCIA-MOLINA, H., AND SPAUSTER, A. 1989. Increasing availability under mutual exclusion constraints with dynamic vote reassignment. *ACM Trans. Comput. Syst. 7*, 4 (Nov.), 394–426.

BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Mass.

BERNSTEIN, P. A., SHIPMAN, D., AND ROTHNIE, J. B. 1980. Concurrency control in a system for distributed databases (SDD-1). *ACM Trans. Datab. Syst. 5*, 1 (Mar.), 18–51.

CAREY, M. J. AND LIVNY, M. 1986. Conflict detection tradeoffs for replicated data. *ACM Trans. Datab. Syst. 16*, 4 (Dec.), 703–746.

CAREY, M. J. AND LIVNY, M. 1988. Distributed concurrency control performance: A study of algorithms, distribution, and replication. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Morgan-Kaufmann, San Mateo Calif., 13–25.

CHEUNG, S. Y., AHAMAD, M., AND AMMAR, M. H. 1990. The grid protocol: a high performance scheme for maintaining replicated data. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. IEEE Computer Society Press, Los Alamitos, Calif., 438–445.

CHEUNG, S. Y., AMMAR, M. H., AND AHAMAD, M. 1991. Multidimensional voting. *ACM Trans. Comput. Syst. 9*, 4 (Nov.), 399–431.

CICIANI, B., DIAS, D. M., AND YU, P. S. 1990. Analysis of replication in distributed database systems. *IEEE Trans. Knowl. Data Eng. 2*, 2 (June), 247–261.

COULOURIS, G., DOLLIMORE, J., AND KINDBERG, T. 2000. *Distributed Systems. Concepts and Design. 3rd edition.* Addison Wesley, Reading, Mass.

Eager, D. L. and Sevcik, K. C. 1983. Achieving robustness in distributed database systems. *ACM Trans. Datab. Syst 8*, 3 (Sept.), 354–381.

Erdös, P. and Lovász, L. 1975. Problems and results on 3-chromatic hypergraphs and some related questions. *Colloq. Math. Soc. János Bolyai 10*, 609–627.

Gifford, D. K. 1979. Weighted Voting for Replicated Data. In *Proceedings of the 7th ACM Symposium on Operating Systems* (Pacific Grove, Calif.). ACM, New York, 150–162.

Gray, J., Helland, P., O'Neil, P., and Shasha, D. 1996. The dangers of replication and a solution. In *Proceedings of the ACM SIGMOD Conference* (Montreal, Ont., Canada). ACM, New York, 173–182.

Gray, J. and Reuter, A. 1993. *Transaction Processing: Concepts and Techniques*. Morgan-Kaufmann Publishers, San Mateo, Calif.

Herlihy, M. 1987. Dynamic quorum adjustment for partitioned data. *ACM Trans. Datab. Syst. 12*, 2 (June), 170–194.

Hwang, S., Lee, K. K. S., and Chin, Y. H. 1996. Data replication in a distributed system: A performance study. In *Proceedings of the 7th Database and Expert Systems Applications* (Zurich, Switzerland). Lecture Notes in Computer Science, vol. 1134. Springer-Verlag, New York, 708–717.

Ingols, K. and Keidar, I. 2001. Availability study of dynamic voting systems. In *Proceedings of the IEEE Internationl Conference on Distributed Computing Systems (ICDCS)* (Phoenix, Ariz.). IEEE Computer Society Press, Los Alamitos. Calif., 247–254.

Jajodia, S. and Mutchler, D. 1990. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Trans. Datab. Syst. 15*, 2 (June), 230–280.

Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G., and Kemme, B. 2001. How to select a replication protocol according to scalability, availability, and communication overhead. In *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS)* (New Orleans, La.). IEEE Computer Society Press, Los Alamitos, Calif., 24–33.

Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G., and Kemme, B. 2002. Scalable database replication middleware. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)* (Vienna, Austria). IEEE Computer Users Society, Los Alamitos, Calif.

Kemme, B. 2000. Database Replication for Clusters of Workstations. Ph.D. dissertation. Dept. of Computer Science, Swiss Federal Institute of Technology Zurich.

Kemme, B. and Alonso, G. 2000. Don't be lazy, be consistent: Postgres-R, A new way to implement Database Replication. In *Proceedings of the IEEE International Conference (VLDB)* (Cairo, Egypt). Morgan-Kaufmann, San Mateo, Calif., pp. 134–143.

Keum, C. S., Choi, W., Hong, E. K., Kim, W. Y., and Whang, Y. 1995. Performance evaluation of replica control algorithms in a locally distributed database system. In *Proceedings of the 4th International Conference on Database Systems for Advanced Applications* (Singapore). World Scientific Press, 388–396.

Kumar, A. 1990. Performance of a Hierarchical Quorum Consensus Algorithm for Replicated Objects. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)* (Paris, France). IEEE Computer Society Press, Los Alamitos, Calif., 378–385.

Kumar, A. 1991. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Trans. Comput. 40*, 9 (Sept.), 996–1004.

Kumar, A. and Segev, A. 1993. Cost and availability tradeoffs in replicated data concurrency control. *ACM Trans. Datab. Syst. 18*, 1 (Mar.), 102–131.

Kumar, A., Rabinovich, M., and Sinha, R. K. 1993. A performance study of general grid structures for replicated data. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)* (Pittsburgh, Pa.), IEEE Computer Society Press, Los Alamitos, Calif., 178–185.

Lewis, P., Bernstein, A., and Kifer, M. 2002. *Databases and Transaction Processing*. Addison-Wesley, Reading, Mass.

Liu, M. L., Agrawal, D., and Abbadi, A. E. 1997. The performance of replica control protocols in the presence of site failures. *Distrib. Syst. Eng. 4*, 2 (June), 59–77.

Lovász, L. 1973. Coverings and colorings in hypergraphs. In *Proceedings of 4th South Eastern Conference on Combinatorics, Graph Theory and Computing*. Utilitas Math., Winnipeg, B.C., Canada, 3–12.

Maekawa, M. 1985. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst. 3*, 2 (May), 145–159.

Naor, M. and Wool, A. 1998. The load, capacity, and availability of quorum systems. *SIAM J. Comput. 27*, 2 (Apr.), 423–447.

Nicola, M. and Jarke, M. 2000. Performance modeling of distributed and replicated databases. *IEEE Trans. Knowl. Data Eng. 12*, 4 (July), 645–672.

Paris, J. F. 1986. Voting with witnesses: A consistency scheme for replicated files. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)* (Cambridge, Mass.). IEEE Computer Society Press, Los Alamitos, Calif., 606–612.

Paris, J. F. 1989. Voting with bystanders. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)* (Newport Beach, Calif.). IEEE Computer Society Press, Los Alamitos, Calif., 394–405.

Paris, J. F. and Long, D. E. 1991. Voting with regenerable volatile witnesses. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)* (Kobe, Japan). IEEE Computer Society Press, Los Alamitos, Calif., 112–119.

Peleg, D. and Wool, A. 1995. The availability of quorum systems. *Inf. Comput. 123,* 2 (Dec.), 210–223.

Peleg, D. and Wool, A. 1997. Crumbling walls: A class of practical and efficient quorum systems. *Distrib. Comput. 10*, 2, 87–97.

Rangarajan, S., Jalote, P., and Tripathi, S. K. 1993. Capacity of voting systems. *IEEE Trans. Softw. Eng. 19*, 7 (July), 698–706.

Rangarajan, S., Setia, S., and Tripathi, S. K. 1995. A fault-tolerant algorithm for replicated data management. *IEEE Trans. Parall. Distrib. Syst. 6*, 12 (Dec.), 1271–1282.

Rothnie, J. B., Bernstein, P. A., Fox, P. A., Goodman, N., et al. 1980. Introduction to a system for distributed databases (SDD-1). *ACM Trans. Datab. Syst. 5*, 1 (Mar.), 1–17.

Saha, D., Rangarajan, S., and Tripathi, S. K. 1996. An analysis of the average message overhead in replica control protocols. *IEEE Trans. Parall. and Distrib. Syst. 7*, 10 (Oct.), 1026–1034.

Spasojevic, M. and Berman, P. 1994. Voting as the optimal static pessimistic scheme for managing replicated data. *IEEE Trans. Parall. Distrib. Syst. 5*, 1 (Jan.), 64–73.

Theel, O. and Pagnia-Koch, H. 1995. General design of grid-based data replication schemes using graphs and a few rules. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)* (Vancouver, B.C., Canada). IEEE Computer Society Press, Los Alamitos, Calif., 395–403.

Thomas, R. H. 1979. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Datab. Syst. 4*, 9 (June), 180–209.

Tong, Z. and Kain, R. Y. 1988. Vote Assignments in Weighted Voting Mechanisms. In *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS)*. (Columbus, Ohio). IEEE Computer Society Press, Los Alamitos, Calif., 138–143.

van Renesse, R. and Tanenbaum, A. S. 1988. Voting with ghosts. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)* (San Jose, Calif.). IEEE Computer Society Press, Los Alamitos, Calif., 456–462.

Weikum, G. and Vossen, G. 2001. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan-Kaufmann Publishers, San Mateo, Calif.

Wool, A. 1996. Quorum systems for distributed control protocols. Ph.D. dissertation, The Weizmann Institute of Science, Rehovot, Israel.

Wool, A. 1998. Quorum systems in replicated databases: Science or fiction? *Data Eng. Bull. 21*, 4 (Dec.), 3–11.

Wu, C. and Belford, G. G. 1992. The triangular lattice protocol: A highly fault tolerant and highly efficient protocol for replicated data. In *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS)*. (Houston, Tex.). IEEE Computer Society Press, Los Alamitos, Calif., 66–73.

Yu, P. S., Dias, D. M., and Lavenberg, S. S. 1993. On the Analytical Modeling of Database Concurrency Control. *J. ACM 40*, 4 (Sept.), 831–872.

Received June 2002; revised February 2003; accepted June 2003