Replication Management in Large Networks

Bernd Freisleben  Hans-Henning Koch  Oliver Thee

Department of Computer Science (FB 20)
University of Darmstadt
D-6100 Darmstadt, Germany

Abstract

The replication of data objects in a large computer network is a difficult task which cannot be approached by simply employing the techniques used in small networks, because the high replication factors possible raise new issues which must be addressed. In this paper, we present a solution to the problem of managing replicas in a large scale environment. The solution is based on a multi-level quorum algorithm for maintaining the consistency of replicas, a probabilistic addressing mechanism for efficiently locating replicas in the system and an efficient scheme for handling dynamic changes in the number of replicas. The feasibility of our approach is demonstrated by presenting performance measurements in a simulated network.

1 Introduction

Since the size of computer networks is likely to grow in the future and a large number of nodes will inherently represent many possible sources of failure, techniques to increase system reliability and availability are of primary importance in a large distributed environment. The replication of critical data at multiple nodes with independent failure modes is one such technique which allows users to continue their work in spite of node crashes and communication link failures. Replication does not only increase the availability of data, but also improves the overall system performance. By storing copies of shared data objects on nodes where they are frequently accessed, communication costs are reduced, because remote requests will no longer be necessary. Unfortunately, it is difficult to achieve high performance and availability while ensuring that the semantics of replicated data objects are identical to those of their non-replicated counterparts. This difficulty arises for the following reasons:

1. The management of replicas requires additional programming effort to provide users transparently with a logical single-copy image of a set of physical copies; without replication transparency the complexity of user applications would increase.

2. The number and placement of replicas are crucial factors for finding a balance between the two diverging goals of maximizing availability and minimizing the cost.

3. The mutual consistency of all replicas must be ensured despite the possible concurrency of conflicting accesses and the presence of node or communication failures, including network partitions.

In a large scale environment with possibly high replication factors efficient solutions to this problem are even harder to find, because laborious failures such as network partitions are more probable and the amount of management overhead, particularly for frequently changing objects, may easily outweigh the benefits of replication. From a performance viewpoint, it does not make much sense to globally replicate frequently changing objects in a large network; high replication factors should consequently only be applied to objects with low update rates. Such fairly static objects are, for example, public information services and commonly used programs which provide functionality needed at a large number of nodes.

In this paper we propose a solution to the problem of managing highly replicated data objects. In particular, we tackle the issues involved in maintaining a minimum but sufficient amount of state information to address and locate replicas in an efficient manner. We show how this state information can accommodate changes in the number of replicas as a result of creating and deleting them. We present a solution to guarantee the mutual consistency of highly replicated data in the presence of concurrency and failures.

The paper is structured as follows. The solution to the problem of maintaining the consistency of replicas is presented in section 2. Section 3 shows how our strategy supports the dynamic creation and deletion of replicas. In section 4 we describe the addressing mechanism. Section 5 presents performance measurements and section 6 concludes the paper and discusses areas for future research.

2 Maintaining the Consistency

Algorithms for maintaining the mutual consistency of replicas require the cooperation of several nodes as opposed to a non-replicated environment where a data item is exclusively controlled by a single node. This cooperation leads to increased costs, since the participating nodes must communicate with each other to ensure that the concurrent execution of operations on replicated data is equivalent to a serial execution on non-replicated objects, a property known as one-copy serializability [3].
The quorum consensus method [11] is a popular algorithm which uses one-copy serializability of read/write operations as its correctness criterion. In this approach each replica of an object is assigned some number of votes. A read quorum of \( r \) votes must be collected for reading and a write quorum of \( w \) votes must be collected for writing. If \( v \) is the total number of votes assigned to a replicated object, then quorums must satisfy two constraints:

1. \( r + w > v \)
2. \( w > v/2 \)

The first constraint prevents read-write conflicts by letting the read and write quorums intersect each other, while the second constraint ensures that two write operations cannot happen in parallel. By assigning the votes and defining the quorums appropriately, the quorum consensus method allows to model other well known approaches to replicated data concurrency control. For example, if each replica is assigned exactly one vote, then in the majority consensus algorithm [21] the majority of votes must be assembled for reading or writing, while the write-all/read-one strategy [3] requires all votes for writing and one vote for reading. Various other voting schemes have been suggested in the literature [6, 13, 19].

The major weakness of the majority-based quorum consensus method is that reading an item becomes fairly expensive, because the number of replicas which must be accessed to collect a read quorum linearly increases with the total number of replicas in the system. Since read operations are assumed to be more frequent than write operations on highly replicated objects, it is essential to keep read quorums as small as possible. However, it is also not desirable to achieve this at the expense of very large write quorums, because the overhead involved in performing write operations would then be almost prohibitive. From a performance viewpoint, the write-all/read-one strategy is therefore not suitable for a large scale environment, and beyond that does not provide an effective solution to network partitioning problems.

In the sequel we present and analyze a scheme for preventing inconsistencies among a large number of replicas which ensures that read operations are cheap without making write operations too expensive. The solution is based on imposing a logical hierarchical structure on the nodes of the network and apply well known flat voting protocols in a layered fashion. The approach of letting algorithms operate on a logical structure is a powerful design methodology which in various forms has been used for other distributed coordination problems [1, 17, 18]. Previous research on employing quorum protocols in a hierarchical manner has been confined to a two-level majority consensus arrangement which has only empirically been evaluated via simulation [15]. In the present context we analytically investigate a particular instance of a multi-level quorum algorithm which is especially suited to satisfy our goals in a large scale networking environment. Before we do so, it is necessary to explain the philosophy behind the multi-level approach.

### 2.1 The Multi-Level Approach

The basic idea of the multi-level proposal is to organize the nodes into a logical hierarchy possibly but not necessarily independent of the physical topology of the network. This organization may in principle be individually set up for each replicated object, but for practical reasons we assume that the hierarchy is set up once for all replicated objects. The leaves of this multi-level tree (level \( l = 0 \)) are the physical nodes and all nodes above the leaf level (\( l > 0 \)) are logical entities which act on behalf of their physical descendants. At the top-level (the root of the tree) the whole network is represented by a single logical node.

Read and write quorums are defined for each level \( l > 0 \). An operation is performed by traversing the tree from the root to the leaf level in order to recursively assemble the required quorums. More precisely, a node on level \( l > 0 \) collects the quorum defined for level \( l \) by gathering the necessary votes among its descendants on level \( l - 1 \). The duties of a logical node are performed by one of its physical descendants. The physical node initiating an operation on a replicated object automatically acts as the logical root node for that operation and the initiating node itself or some other physical nodes may take the responsibility for the coordinator functions of the logical nodes between the root and the leaves. When the leaf level is reached, the replicas stored on the physical nodes are locked and the votes (together with each replica's version number and node id) are returned to the coordinators until they arrive at the root of the tree. Fault tolerance against initiator/coordinator crashes and communication failures may be achieved by employing the standard election and commit protocols described in [4] within each group (the term group is used as a synonym for logical node throughout the paper). A correctness proof for the multi-level quorum method is given in [15].

To illustrate the approach, consider a system with 9 replicas of an object stored on 9 physical nodes which are organized into 3 groups with 3 nodes each as shown in figure 1. Assume that each replica has exactly one vote.

![Figure 1: An example of a two-level hierarchy with 9 physical nodes](image)

If the write-all/read-one strategy (WAS) is used on level \( l = 2 \) and the majority consensus strategy (MCS) on level \( l = 1 \), then the WAS requires for a read operation to consult only one logical node on level 1, whereas all logical nodes on level 1 have to be consulted for a write operation. The MCS on level 1 requires the majority of votes among the nodes on level 0 within each group they belong to. In this example the total number of votes required for a read operation is 2, for a write operation it is 6. The “flat” WAS requires 1 vote for reading and 9 votes for writing.
whereas the "flat" MCS requires 5 votes for performing any operation. Note that both flat approaches are special cases of the general multi-level method by simply introducing one logical node on level \( l = 1 \) which directly represents all physical nodes.

The particular two-level arrangement with the WAS applied on top of the MCS is well suited for an application environment where the read costs have to be kept low without increasing the write costs too much. We refer to this arrangement as the electoral district strategy (EDS), since it somewhat resembles the procedure of organizing the voting population of a country into districts and determine the winning candidate of each district who in turn represents the election result in higher level political committees. In the next section we compare the EDS with the MCS and the WAS.

2.2 Analysis

Although the multi-level approach allows to organize the nodes into an arbitrary number of levels, each with a possibly different replication algorithm, this paper is only concerned with the EDS. Other two-level and three-level strategies employing different combinations of the MCS and the WAS are considered in [10].

In the sequel the EDS is analyzed with respect to the message costs associated with it and the degree of availability it provides. We consider only one particular object with its replicas and show how an operation is performed within the EDS two-level arrangement. The generalization to an arbitrary number of replicated objects is straightforward.

In order to keep the analysis simple, we assume that in each group the number of nodes on level \( l = 0 \) is \( n^{(0)} \) where \( n^{(0)} \) is odd and that there are \( n^{(1)} \) logical nodes on level \( l = 1 \), i.e. \( N = n^{(0)} n^{(1)} \), where \( N \) is the total number of nodes. The following analysis is based on the assumption that each node has a replica of an object.

### 2.2.1 Costs

Let \( S \) be a voting-strategy based on read and write operations. Then \( \text{MinRN}_{S} \) (\( \text{MinWN}_{S} \)) is the minimum number of replicas which must be accessed using strategy \( S \) in order to collect a read-quorum (write-quorum).

Figure 2 shows \( \text{MinRN} \) and \( \text{MinWN} \) for the EDS, MCS and WAS.

<table>
<thead>
<tr>
<th></th>
<th>EDS</th>
<th>MCS</th>
<th>WAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinRN</td>
<td>( \left[ \frac{n^{(0)}+1}{2} \right] \cdot 1 )</td>
<td>( \left[ \frac{N}{2} + 1 \right] )</td>
<td>1</td>
</tr>
<tr>
<td>MinWN</td>
<td>( \left[ \frac{n^{(0)}+1}{2} \right] \cdot n^{(1)} )</td>
<td>( \left[ \frac{N}{2} + 1 \right] )</td>
<td>( N )</td>
</tr>
</tbody>
</table>

Figure 2: Minimal costs of EDS, MCS and WAS

The formulas presented above have been used to calculate the costs of the EDS in a network with approximately 200 nodes and different group sizes (see figure 3).

<table>
<thead>
<tr>
<th>( N/n^{(0)} )</th>
<th>( n^{(1)} )</th>
<th>( \text{MinRN}_{EDS} )</th>
<th>( \text{MinWN}_{EDS} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>1</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>201</td>
<td>3</td>
<td>67</td>
<td>2</td>
</tr>
<tr>
<td>200</td>
<td>5</td>
<td>40</td>
<td>3</td>
</tr>
<tr>
<td>203</td>
<td>7</td>
<td>29</td>
<td>4</td>
</tr>
<tr>
<td>207</td>
<td>9</td>
<td>23</td>
<td>5</td>
</tr>
<tr>
<td>209</td>
<td>11</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>208</td>
<td>13</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>210</td>
<td>15</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>204</td>
<td>17</td>
<td>12</td>
<td>9</td>
</tr>
</tbody>
</table>
| ... | ... | ... | ... | ...
| 201 | 201 | 1   | 101 | 101 |

Figure 3: Costs of the EDS with varied group sizes

Although the definition of the EDS allows each group to have a different number of members, we again assume that \( N = n^{(1)} \cdot n^{(0)} \). The costs of the MCS are equal to those of the EDS with \( n^{(0)} = 201 \) and the costs of the WAS are equal to those with \( n^{(0)} = 1 \).

It is easy to see that a variation of the group sizes allows to adjust the costs of the operations, since the EDS can adopt the whole range of values given in figure 3. If, for example, the network on level 1 is organized into \( n^{(1)} = 67 \) groups each containing \( n^{(0)} = 3 \) nodes, a read operation requires to consult 2 nodes, whereas the MCS requires to access at least 101 nodes. As explained before, write operations in the EDS are more expensive than in the MCS. If the group size is again \( n^{(0)} = 3 \), then the EDS requires at least 134 consultations whereas in the MCS only 101 nodes have to be accessed.

If all nodes have exactly one vote, then the following holds for the MCS (and also for the WAS):

\[
\text{MinRN}_{MCS} + \text{MinWN}_{MCS} > N \tag{1}
\]

\[
\text{MinWN}_{MCS} > \frac{N}{2} \tag{2}
\]

In the case of the EDS the first inequality is only valid if \( n^{(1)} = 1 \) or \( n^{(1)} = N \), otherwise the following holds:

\[
\text{MinRN}_{EDS} + \text{MinWN}_{EDS} = \left[ \frac{n^{(0)}+1}{2} \right] + n^{(1)} \left[ \frac{n^{(0)}+1}{2} \right] \leq N \quad \forall n^{(1)} = 2, \ldots, N - 1 \tag{3}
\]

This demonstrates that in the EDS the read and write quorums do not linearly depend on each other. It can be seen from figure 3 that the number of replicas required for performing the read operation increases linearly with \( n^{(0)} \), but the cost of a write operation decreases faster than linearly as \( n^{(0)} \) increases. Any single-level quorum strategy does not have this property, because their correctness is based on the fact that
any decrease of the read quorum must result in a proportional increase of the write quorum. In general, the following holds:

\[ \text{MinRN}_{WAS} < \text{MinRN}_{EDS} < \text{MinRN}_{MCS} \] (4)

for \( 1 < n^{(1)} < N \land n^{(1)} \cdot n^{(0)} = N \land n^{(1)}, n^{(0)} \in \mathbb{N} \)

and

\[ \text{MinRN}_{WAS} > \text{MinRN}_{EDS} \geq \text{MinRN}_{MCS} \] (5)

for \( 1 < n^{(1)} < N \land n^{(1)} \cdot n^{(0)} = N \land n^{(1)}, n^{(0)} \in \mathbb{N} \)

### 2.2.2 Availability

In order to perform an operation on a replicated object successfully, it is required in all strategies to reach a sufficient number of replicas in order to satisfy the quorums. Since it sometimes may be impossible to reach particular replicas due to node crashes or communication failures, the probability of successfully performing a read or a write operation is an important criterion for determining the degree of availability offered by a replication strategy. In this section we analyze the degree of availability provided by the different strategies. Again we assume that each node has a replica.

We first need some definitions:

Let \( p \) be the probability that the replica at an arbitrary node can be reached at a certain time and will remain reachable until a read or write operation is finished.

Let \( p\{\text{Read}\} \) (\( p\{\text{Write}\} \)) be the probability that a read (write) operation can successfully be performed and let \( p^{(1)} \) \( (p^{(2)}) \) denote the probability that the quorum on level 1 for a read (write) operation can be collected.

Since these probabilities follow a binomial distribution (a node is either up or down), they are easy to calculate. Figures 4a and 4b show the results.

<table>
<thead>
<tr>
<th></th>
<th>WAS</th>
<th>EDS</th>
</tr>
</thead>
</table>
| \( p\{\text{Read}\} \) | 1 - (1 - \( p \))^N | \( p_R^{(1)} \) | \[
\sum_{i=0}^{n^{(0)}} \binom{n^{(0)}}{i} p^i (1 - p)^{n^{(0)} - i}
\] |
| \( p\{\text{Write}\} \)  | \( p^N \)   | \( p_W^{(1)} \) | \[
\sum_{i=0}^{n^{(0)}} \binom{n^{(0)}}{i} p^i (1 - p)^{n^{(0)} - i}
\] |

\[ p\{\text{Read}\} \]
\[
1 - (1 - p^{(1)})^{n^{(1)}}
\]

\[ p\{\text{Write}\} \]
\[
(p^{(2)})^{n^{(1)}}
\]

Figure 4b: Availability of the EDS

It is easy to see that the variation of the group sizes also takes effect on the availability of the operations. Figure 5 and figure 6 show the availabilities of the EDS for values of \( p \) ranging from 0 to 1 in a network with approximately 200 nodes and different group sizes.

Figure 5: Availability of the read operation for varied group sizes \( n^{(0)} \)

Figure 6: Availability of the write operation for varied group sizes \( n^{(0)} \)

It is not surprising that the availability of the read operation is best when the group sizes \( n^{(0)} \) are small, i.e. when the read costs are low. The write availability increases as the group sizes get larger, but then the read operation becomes more expensive.
The two figures can be used to determine the most favourable group assignment. For instance, in figure 6 the curve of a given probability \( p \) (that a node is reachable) may be followed until a desired write availability is reached and the optimal group size \( n_{(0)} \) is then obtained at the point of intersection. This group size in turn provides the cheapest read operation possible with these parameters.

A comparison of the availabilities provided by the EDS, MCS and WAS reveals that for a fixed \( p \in (0, 1) \) the following holds:

\[
p\{\text{Read}_{\text{AS}}\} > p\{\text{Read}_{\text{EDS}}\} > p\{\text{Read}_{\text{MCS}}\}
\]
for \( 1 < n^{(1)} < N \land n^{(1)} \cdot n^{(0)} = N \land n^{(1)}, n^{(0)} \in \mathbb{N} \)

and

\[
p\{\text{Write}_{\text{AS}}\} < p\{\text{Write}_{\text{EDS}}\} < p\{\text{Write}_{\text{MCS}}\}
\]
for \( 1 < n^{(1)} < N \land n^{(1)} \cdot n^{(0)} = N \land n^{(1)}, n^{(0)} \in \mathbb{N} \)

As expected, the availabilities of read and write operations in the EDS are between those of the WAS and MCS. Thus, it is possible to both provide cheap read operations and write operations with an acceptable degree of availability.

### 3 Number and Placement of Replicas

In addition to the problem of maintaining the consistency of replicated data, the number and placement of replicas in the network deserve special attention. In general, increasing the number of replicas results in higher availability. On the other hand, a higher number of replicas incurs higher storage costs and also higher processing costs if the object changes on a frequent basis. The number and placement of replicas should consequently depend on the access probabilities at the different nodes and the characteristics of the network. This implies that there is no a priori optimal strategy for assigning replicas to the nodes in the network [14]. It is definitely wrong to shift such a problem to the system administrators, because the large number of factors necessary to take into consideration would inevitably lead to dissatisfactory performance results. This problem should consequently be automatically handled by the system.

An adaptive solution to the replica assignment problem has been presented in [9]. The basic idea of this solution is that replicas are dynamically created or deleted by the system whenever circumstances dictate. There are mechanisms which observe and record the number and type of accesses to a replicated object and use this information to assess the current replica assignment scenario according to various cost functions. If certain predefined thresholds are reached, additional replicas are created or some are deleted, depending on the expected cost savings in relation to the amount of communication overhead required.

In the context of our present work we assume that such a mechanism is available, which implies that the replication protocol used must be capable of dealing with dynamic creations and deletions of replicas. Since in a voting strategy the assignment of the votes and the determination of the required quorums is based on a fixed number of replicas, it is necessary to either dynamically adjust the number of votes or the quorum sizes when the number of replicas changes. Both approaches have been proposed to increase the availability of replicated data in the presence of network partitioning failures [3, 5, 12].

Whatever type of adjustment is employed, it is essential to keep the number of nodes which must be informed about the change small, because otherwise the vote or quorum reassignment protocol is susceptible to severe performance penalties due to the large amount of communication overhead involved. In a multi-level organization this is almost natural to achieve, since the scope of the reassignment procedure can be restricted to the nodes in each group. The basic idea of our approach is to ensure that the quorum sizes and the sum of all votes in each group are independent of the total number of replicas in the system. This means that the sum of all votes and the required quorums in each group remain constant.

The reassignment algorithm we propose assures that any majority of nodes with replicas in a group always owns a majority of votes. This prevents that a single replica can have the majority of the votes alone (except for the case where there are only one or two replicas in the group). In addition, when there are only two partitions of a group, the algorithm does not permit the two partitions to have an equal number of votes.

We assume that the number of nodes in a group is \( n \) which is known to each of its members. Then the total sum of votes within this group is given by

\[
\upsilon_{\text{sum}} = \prod_{i=1}^{\lfloor \frac{n}{2} \rfloor} (2i - 1) \quad (8)
\]

Thus, \( \upsilon_{\text{sum}} \) just consists of odd factors and therefore is always odd. It can now easily be distributed among any number of replicas up to \( n \).

If the number of replicas \( r \) is odd, then each replica gets \( v \) votes, where

\[
v = \frac{1}{r} \cdot \upsilon_{\text{sum}} \quad (9)
\]

If \( r \) is even, then \( r - 1 \) replicas each obtain \( v \) votes, with

\[
v = \frac{1}{r + 1} \cdot \upsilon_{\text{sum}} \quad (10)
\]

and the remaining replica gets

\[
v = \frac{2}{r + 1} \cdot \upsilon_{\text{sum}} \quad (11)
\]

votes. Nodes without a replica do not get any votes (\( v = 0 \)).
In order to illustrate our reassignment algorithm, we now present two examples for group sizes \( n = 3 \) and \( n = 4 \).

For \( n = 3 \), \( v_{\text{sum}} \) computes to

\[
v_{\text{sum}} = 1 \cdot 3 = 3
\]

and for \( n = 4 \) we obtain

\[
v_{\text{sum}} = 1 \cdot 3 \cdot 5 = 15.
\]

The resulting distributions of votes \( v_i \) to the nodes of the groups for different numbers \( r \) of replicas are presented in figure 7.

<table>
<thead>
<tr>
<th>( n = 3 )</th>
<th>( n = 4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r )</td>
<td>( v_1 )</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 7: Vote reassignment for group sizes \( n = 3 \) and \( n = 4 \)

Note that for an even number of replicas one of them is slightly favoured against the others, but as explained above, this is our "tie-breaker" when the group partitions into two subgroups with an equal number of replicas. Since the vote reassignment is done in relatively small groups compared with the total number of nodes in the network, we believe that our approach is especially suited for a highly replicated environment.

4 Address Management

One of the main issues involved in delivering a single-copy image to application programs is the provision of a sufficient amount of state information to transparently address and locate replicas in an efficient manner. Depending on the nature of the operations to be performed and the mechanisms to implement them, a node must be able to locate one, a subset, or all replicas. The problem of locating replicas is a special case of a general naming service [16, 20] for a distributed system. Whereas naming can usually be seen as a one-to-one mapping between an object's identification and its physical location, replicas require a one-to-many mapping to associate an object's name to a set of locations.

There are several possible approaches for providing a node with the necessary information to locate replicas. They may be briefly summarized as follows.

- **Centralized Information**
  In this approach the mapping information for all replicated objects in the system is stored at a known server node. The location of replicas can easily be obtained by questioning the server. The amount of information at each node is restricted to the address of the server. However, a single central component might not only become a bottleneck, but also would seriously affect system operation in case of a crash. Furthermore, in a large geographically dispersed network, a single server will be inefficient owing to the long delays in accessing it.

- **Distributed Information**
  In this approach the complete mapping information for all replicated objects is stored redundantly at all nodes. Although this organization avoids the disadvantages of the centralized server approach, it requires a great deal of additional work to maintain the consistency of the replicated server information. This implies that any change in the mapping information has to be propagated to all nodes, which is far too costly in a very large network. Furthermore, a substantial amount of memory space is needed at each node for storing the mapping.

- **Hierarchical Information**
  In this approach the system is logically decomposed into several sets of nodes, each with its own server storing only mapping information relevant to the corresponding set. These sets are hierarchically organized into one or more additional groupings, such that each higher layer server has just enough information to direct enquires to one of its subservers containing the full information. This hierarchical server approach is very attractive for a large scale environment, because it limits the amount of information needed at every node in a reasonable manner [8]. However, if the nodes in a set only know about exactly one server, problems arise if this server crashes.

Our solution to the addressing problem is a hybrid approach which incorporates several features of all the above approaches. The proposed mechanism is completely distributed, but uses information about the multi-level organization to provide a starting point for finding replicas and repeatedly questions further nodes to effectively locate all replicas required.

The basic idea is that each node has a catalogue containing probabilistic information in the sense that it lists all the nodes which with a certain probability are assumed to have a replica of the object. Additionally, there are version numbers as part of each entry associated with each replica in order to determine the most up-to-date replicas. The structure of the catalogue reflects the multi-level organization of the network and the stored information need not be consistent with the catalogues of other nodes.

Based on the assumption that each node knows all the other nodes in the network, the locations of the
necessary replicas can be determined by first checking the nodes contained in the catalogue. If this should not be sufficient to collect the required quorum, the remaining nodes not listed in the catalogue are repeatedly asked to gather the required quorum. The basic structure of a catalogue is shown in Figure 8.

We now present an example to illustrate how the address management coherently cooperates with the previously described mechanism to support the dynamic creation and deletion of replicas. The example shown in Figure 8 is based on the simplified, flat structure instead of the complete structural information shown in Figure 8. Each node knows that there are four nodes $K_1, \ldots, K_4$ in the network and that the total sum of votes is 15. The nodes $K_1$ and $K_3$ have a replica, because their votes are greater than zero.

Figure 9 shows four nodes with their associated votes and catalogues at a certain point of time. Each node knows that there are four nodes $K_1, \ldots, K_4$ in the network and that the total sum of votes is 15. The nodes $K_1$ and $K_3$ have a replica, because their votes are greater than zero.

The knowledge of the individual nodes about the location of replicas is represented in the different catalogues: for example, $K_1$ "thinks" that $K_1$ and $K_3$ have a replica, whereas $K_2$ thinks that only $K_3$ has a replica of the object. In this sense the address management is probabilistic. The information in the catalogues does not need to be consistent which is very desirable in a large networking environment, especially when replicas are dynamically created and deleted. Although it is useful to inform several nodes about changes of the location of replicas, it is not necessary to inform any. Therefore, from the view of the address management the creation and deletion of replicas can be done without the help of any other node.

Suppose $K_2$ needs to find out the location of all replicas in the system. The first action is to call the nodes in the order listed in its catalogue, which in this case is only $K_3$. $K_3$ answers the call by positively acknowledging the existence of a replica and submitting its votes (votes=5). Since there are no further entries in the catalogue, $K_2$ starts successively sending enquiries to all the remaining nodes in the network. If the order of these enquiries is $K_1, K_2, K_4$ the search for replicas can already be stopped after having asked $K_1$, because the transferred votes are equal to the total sum of votes in the system. The need to "ask all the remaining nodes" not contained in the catalogue reduces constantly during system operation, because each node "learns" about the location of replicas whenever an operation on them is performed.

This basic mechanism may be improved by additionally sending the catalogue of a called node to the invoker and by using the catalogue information to update the catalogue of the invoking node. By exploiting the multi-level organization the communication between nodes for locating replicas can be reduced. After having performed a creation or deletion operation on a replica the node informs all of its group members. Each node has correct information about whether it holds a replica and whether any other member of its group does. This requires no communication. Thus, each node has best (but not necessarily accurate) information about its own group. If a node initiates an operation which requires the cooperation with other groups of nodes, it dynamically chooses an arbitrary node within each group. We refer to such a node as the group coordinator. Any group coordinator takes care of the various actions which have to be performed within its group and informs the initiating node about the result. Therefore, the initiating node does not need to maintain information about the location of replicas of other groups. It is simply sufficient to find group coordinators in order to put them in charge of executing the desired operation.

5 Performance Measurements

In order to investigate the cooperation of the presented concepts for replication management, we implemented them in a simulated network on a VAX-based station. The simulated network is able to model node and communication failures, including network partitions. The implementation was written in the strongly-typed object-oriented programming language FLELIS [17].

By defining the number of nodes and their organization into hierarchically structured groupings, we simulated the EDS, MCS and WAS strategies. A graphical user interface is provided which allows to monitor the behaviour of the three strategies in a convenient manner.

In the sequel we compare the performance of the three strategies. In order to keep the graphical representation of the network simple, the measurements are based on a network consisting of 10 nodes, but our analytical investigation indicates that similar results are to be expected for a large network. In the EDS, the nodes 1, 2 and 3 are in group 1, the nodes 4, 5, and 6 are in group 2, and the nodes 7, 8, 9, and 10 are in group 3.

The measured execution time of an operation includes the time needed for creating and deleting processes, the time for acquiring and releasing locks on the nodes involved, the time for transferring the messages between the nodes via the simulated network, various local table lookups as well as updating the address information and the graphical representation on the screen. The comparison is based on the
mean execution time of 100 runs. Since the absolute values obtained by this somewhat crude form of measurement are not interesting per se (the system clock was started when the execution of an operation began and was stopped after it terminated), we have set the value of the worst strategy in each experiment to 100% and consequently present percentage values for the other strategies showing their performance in relation to the worst one.

Figure 10 shows the results of performing an operation in a network where a replica of an object resides at each node. The obtained results confirm our analytic investigations: compared with the MCS, the EDS offers low-cost read operations on replicated objects. As expected, the cost of a write operation performed at node 1 is the lowest among the three when the MCS is used and is only slightly higher when the EDS is employed.

As expected, the cost of a write operation performed at node 1 is the lowest among the three when the MCS is used and is only slightly higher when the EDS is employed.

All nodes have a replica of an object

<table>
<thead>
<tr>
<th>%</th>
<th>EDS</th>
<th>MCS</th>
<th>WAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 1 reads the object</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Node 1 writes the object</td>
<td>32.6</td>
<td>16.2</td>
<td>45.6</td>
</tr>
</tbody>
</table>

Figure 10: Performance of read and write operations for 10 replicas

Figure 11 shows the performance results when an operation is performed in a network where only the nodes 1, 2, 4, 5 and 7 have a replica of the object. The results are qualitatively similar to those obtained in a network where a replica is stored on each node.

Nodes 1, 2, 4, 5, and 7 have a replica of an object

<table>
<thead>
<tr>
<th>%</th>
<th>EDS</th>
<th>MCS</th>
<th>WAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 2 reads the object</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Node 2 writes the object</td>
<td>62.6</td>
<td>35.8</td>
<td>34.0</td>
</tr>
</tbody>
</table>

Figure 11: Performance of read and write operations for 5 replicas

Although additional performance measurements are necessary to demonstrate the suitability of the EDS for large networks, we think that the results obtained already indicate the advantages the EDS offers for managing highly replicated objects in large distributed systems even when the number of replicas of an object changes dynamically.

6 Conclusions

In this paper we have investigated the issues involved in managing replicated objects in a large-scale networking environment. Our solution is characterized as follows:

In order to master the complexity of a very large system, we logically organize the nodes of the network into hierarchically structured groupings. A simple probabilistic addressing mechanism which easily adapts to dynamic configuration changes has been proposed to address and locate replicas of an object in an efficient manner.

A suitable mechanism for dealing with the consistency of replicas in the presence of network partitions...
has been presented. It is a particular instance of the class of multi-level quorum strategies. The approach is immune to changes in the number of participating nodes as compared with conventional flat approaches and ensures that read operations, which are far more frequent than write operations on highly replicated objects, are cheap.

The presented concepts form the basis of an implementation which is currently carried out to provide reliable network information services, such as network address and password files, in our campus network consisting of UNIX workstations.

There are a number of further research issues which we plan to address in the future, such as analytically investigating strategies with an arbitrary number of levels, using other voting protocols like dynamic voting [13] or voting with witnesses [19] in a multi-level environment, employing different strategies in each group on each level and extending the basic multi-level voting approach to use type-specific information of objects instead of simple read/write-semantics.

References