Embedding Protocols for Scalable Replication Management[1]

Henning Koch

Dept. of Computer Science
University of Darmstadt
Alexanderstr. 10
D-64283 Darmstadt
Germany
koch@isa.informatik.th-darmstadt.de

Keywords:

distributed systems, availability, data replication, logical structures, voting protocols

[1]This work was done when the author was with Digital Equipment's CEC Karlsruhe, Germany. It was partially supported by the German Ministry of Research and Technology under grant no. 125-3590-17F.

Summary

The concept of embedding protocols is proposed as a new technique for constructing replication protocols. A number of replicas is combined to one logical building block and a replication protocol is applied to it. By allowing those building blocks to become part of another building block a hierarchy is set up. Since every building block can use a different replication protocol, the resulting protocol can be regarded as an embedding replication protocol. It is shown how these protocols can be constructed and analyzed.

1 Introduction

In the literature there exists a variety of different replication protocols. These protocols are designed to either increase the availability of data items in a distributed environment [14, 8] or decrease the access costs [13]. A precondition for achieving decreased communication costs is that only one copy of the replicated data has to be accessed and that this copy is local or at least located at a nearby node. However, this can only be achieved for read operations. To preserve consistency write operations then have to update all copies which is very expensive and results in a very poor write availability. Some protocols [13] weaken this constraint and let write operations succeed by only updating as many copies as reachable, but at least one copy. As a consequence, write operations can endanger the consistency of the replicated object during the occurrence of network partitions because simultaneous writes are allowed in all partitions. Other replication protocols try to provide highly available access operations while at the same time ensuring one-copy serializability [4]. An example for this is the Majority Consensus Protocol [14] which has equally high read and write availabilities but has the drawback of unacceptable high read costs. Based on this protocol several improvements have been proposed: The Weighted Voting Protocol [7] allows a reduction of the read costs but at the expense of a more expensive and less available write operation. For this protocol, the read and the write costs linearly depend on the total number of replicas. Multi-Level Voting [6] and Hierarchical Quorum Consensus [10] organize the replicas in a hollow logical tree structure. By using this organization, the operation costs can be reduced to $O\left[n^{0.63}\right]$ while the operation availabilities are still sufficiently high. The Grid Protocol [5] uses a logical grid structure as an organization for the replicas. Read operations have to collect a *c-cover*, i.e. one replica of every grid column, while write operations must additionally access all replicas of an arbitrary column. By exploiting the grid structure it is possible to reduce the operation costs to $O\left[\sqrt{n}\right]$ and still having acceptable operation availabilities. In the Tree Quorum Protocol [2] and the Logarithmic Protocol [9] the replicas are organized in a logical tree structure. These protocols allow read operations to

succeed by only reading a single replica. However, the replicas are not treated equally: the closer a replica is located to the root replica of the tree the more important is it. Disadvantages of both protocols are their poor write availabilities and the fact that the root replica is a potential bottleneck.

All these replication protocols are compromising operation costs and availabilities. By arranging these protocols in a hierarchy, it is possible to design new protocols which inherit the advantages of the embedded strategies. We call a protocol which consists of a hierarchy of protocols an *embedding* protocol. Multi-Level Voting consists of a hierarchy of Weighted Voting Protocols and the Hierarchical Grid Protocol [11] consists of a hierarchy of Grid Protocols. Both protocols are special cases of embedding protocols since they use a hierarchy but incorporate only a single type of protocol. The idea of using different replication protocols within a hierarchy also occurred in [1, 9].

The rest of the paper is organized as follows: In section 2, we introduce the model and give an example for an embedding replication protocol. A method for analyzing embedding replication protocols is proposed in section 3. We present a correctness criterion in section 4 and exploit it for constructing an advanced algorithm for a class of embedding replication protocols. Section 5 concludes the paper with a summary and a discussion of the future work.

2 Embedding Replication Protocols

In order to set up a hierarchy with replication protocols we first define the building block of the hierarchy: a *logical replica*. A logical replica provides a one-copy image of a replicated object. It offers well defined access operations and it is hiding its internal implementation. A logical replica consists of a group of replicas (called its *members*), which are organized in a logical structure, and defined strategies which are responsible for preserving the consistency. The structure defined for the replicas can be as simple as a set or more complex as a tree or a grid structure. For every access mode we define a consistency strategy which is depending

on the used structure, e.g. for a set we define the consistency strategies `one`, `all`, `majority`, `quorum`, etc. For example, choosing `one` for the read operation implies that one replica has to be accessed to read the object. Consequently, with the consistency strategy pair of `one` for the read and `all` for the write operation we can model the Read-One/Write-All strategy [3].

If a logical replica itself contains logical replicas as members then a hierarchy is set up. In contrast to the logical replicas we define the *real replicas* as the leaves of the hierarchy. In the real replicas an instance of the replicated object's value is stored physically while logical replicas have to compute it from their members' value.

The technique of using logical replicas leads to a number of advantages. In order to compose a logical replica, it is possible to group together real replicas which are stored on the nodes of a single subnetwork. Thus, these replicas can communicate by mostly using the cheap communication links within the subnetwork. Furthermore, special subnetwork characteristics can be exploited: if the subnetwork is e.g. an ethernet then multicasts can be used for communication and the Bystander method [12] can be used, if the subnetwork is partition-free. For each subnetwork it is possible to install logical replicas that suit best to it. By combining these logical replicas to an embedding replication protocol, it is possible to design a tailor-made replication protocol with optimal costs and availabilities for any given network.

The required access operations for a logical replica are given below:

- `lock(IN: replica, mode;`

  `OUT: return_code, value, version)`

  A logical replica is locked in the specified mode. If the locking is successful then its current value is delivered. Locking a logical replica is done by locking its members. A lock operation on a logical replica is successful if enough members can be successfully locked to satisfy the consistency strategy with respect to the defined structure.

- `unlock(IN: replica, mode)`

This operation unlocks a previously locked logical replica. This is done by unlocking all locked members.

- `write&unlock(IN: replica, new_value)`

  The value of a previously locked logical replica is updated and the replica is unlocked afterwards. Writing a logical replica's value is done by writing the value of all of its previously locked members. In order to maintain the consistency of the replicated object even in the presence of failures, the write operation has to be performed by using an atomic commit protocol [4].

All operations performed on a replicated object have to lock its root replica. If we assume that this logical replica is stored on a single node, then no operation can succeed if this node is down. Similar statements hold for the other logical replicas. For this reason, copies of each logical replica are distributed among a number of nodes and managed by using the Read-One/Write-All algorithm. This algorithm allows to retrieve the information stored in a logical replica (members, structure and consistency strategies) by accessing a single copy of it, preferably a local one. Write operations on logical replicas, which are only needed to change this information, are performed rarely. However, this does not imply that the value of the embedding replicated object is managed with the Read-One/Write-All algorithm.

In the following we give an example of how to use the model.

2.1 An Example Embedding Protocol

Figure 1 shows an example object which is replicated by using an embedding protocol with five logical replicas $R_1, \ldots, R_5$ and fifteen real replicas $R_6, \ldots, R_{20}$.

We call a logical replica which uses a set structure for its members a *logical set replica*. Analogously, a *logical grid replica* is a logical replica with members organized in a grid structure. Using these definitions, R_1 and R_3 are logical grid replicas and R_2, R_4, and R_5 are logical set

replicas.

Figure 2 lists for each logical replica the defined structures and their consistency strategies for the read and the write operation.

For the strategy `c-cover` one replica of each column of the grid has to be locked while for `column&c-cover` additionally locking an arbitrary full column of the grid is necessary [5]. The strategy `quorum(`k`)` needs locks on at least k replicas of the set.

Assume that there is a read request for the embedding replicated object, initiated at an arbitrary node. The call is re-directed to the logical root replica R_1 which is locked in read mode:
`lock(IN:` R_1`, 'read'; OUT:` rc_1`,` X_1`,` VN_1`).`

R_1 organizes its members as a grid. In order to maintain consistency, a read operation must lock a c-cover, e.g. R_2 and R_4. Thus, the generated calls are
`lock(IN:` R_2`, 'read'; OUT:` rc_2`,` X_2`,` VN_2`)` and
`lock(IN:` R_4`, 'read'; OUT:` rc_4`,` X_4`,` VN_4`).`

A set structure is used for the members of both R_2 and R_4. Let R_2 lock the real replicas R_6 and R_8 which is sufficient to satisfy the consistency strategy `majority` and let R_4 lock the real replica R_{15} to fulfil the `one` strategy. Thus, the calls
`lock(IN:` R_6`, 'read'; OUT:` rc_6`,` X_6`,` VN_6`),`
`lock(IN:` R_8`, 'read'; OUT:` rc_8`,` X_8`,` VN_8`)`
and
`lock(IN:` R_{15}`, 'read'; OUT:` rc_{15}`,` X_{15}`,` VN_{15}`)`
are generated. R_6,...,R_{20} are real replicas which are simply locked and subsequently deliver their value and version number to the caller. We assume that all replicas are available and can be successfully locked. Let
`(`rc_6`=OK,` X_6`=42,` VN_6`=13)`
be the return value for R_6,

(rc$_8$=OK, X$_8$=69, VN$_8$=12)

the one for R_8, and

(rc$_{15}$=OK, X$_{15}$=61, VN$_{15}$=11)

the one for R_{15}. Each caller chooses the most actual value and delivers it together with the corresponding version number to its own caller. R_1 receives

(rc$_2$=OK, X$_2$=42, VN$_2$=13)

from R_2 and

(rc$_4$=OK, X$_4$=61, VN$_4$=11)

from R_4 and returns

(rc$_1$=OK, X$_1$=42, VN$_1$=13)

as the current value and version of the embedding replicated object.

Figure 3 shows the locks hold on the real replicas. Locked replicas are marked with an "L".

3 Analysis of Embedding Replication Protocols

In this section, we define a cost and an availability measure for embedding strategies that enable us to compare different strategies. For the analysis, we assume that each node owns exactly one real replica and a copy of every logical replica.

3.1 Costs

We define the operation costs of an embedding protocol as the number of real replicas that have to be locked in order to perform the desired operation. Since in an embedding strategy all logical replicas of a particular hierarchy level can differ, e.g. in the defined structure, we make the assumption that the operation chooses each logical replica with the same probability and define an average value as cost measure. The costs $c_{op}(r)$ of operation *op* executed on replica r are

$$c_{op}(r) = \begin{cases} 1 & \text{if } r \text{ is a real replica} \\ \sum_{S \in M_{op}(r)} \frac{\sum_{s \in S} c_{op}(s)}{|M_{op}(r)|} & \text{else} \end{cases} \quad (1)$$

where $M_{op}(r)$ denotes the set of *minimal* sets of replicas which satisfy the consistency strategy defined for replica r and operation *op*.

Applied to the example in section 2.1 we compute the costs of a write operation (w) to

$$\begin{aligned} \mathrm{c}_w(\mathrm{R}_1) &= \tfrac{1}{4}\left((\mathrm{c}_w(\mathrm{R}_2) + \mathrm{c}_w(\mathrm{R}_4) + \mathrm{c}_w(\mathrm{R}_5)) + \right. \\ &\quad (\mathrm{c}_w(\mathrm{R}_3) + \mathrm{c}_w(\mathrm{R}_4) + \mathrm{c}_w(\mathrm{R}_5)) + \\ &\quad (\mathrm{c}_w(\mathrm{R}_2) + \mathrm{c}_w(\mathrm{R}_3) + \mathrm{c}_w(\mathrm{R}_4)) + \\ &\quad \left.(\mathrm{c}_w(\mathrm{R}_2) + \mathrm{c}_w(\mathrm{R}_3) + \mathrm{c}_w(\mathrm{R}_5))\right) \\ &= \tfrac{1}{4}\left((2+3+4) + (3+3+4) + \right. \\ &\quad \left.(2+3+3) + (2+3+4)\right) = \mathbf{9} \end{aligned}$$

3.2 Availability

For the availability analysis we make the additional assumptions that node failures occur independently with a probability $(1-p)$ and that communication links do not fail.

We compute the availability $\wp_{op}(r)$ of an operation *op* executed at a replica r by

$$\wp_{op}(r) = \begin{cases} p & \text{if } r \text{ is a real replica} \\ \sum_{S \in A_{op}(r)} \prod_{s \in S} \wp_{op}(s) \prod_{t \in (R \setminus S)} (1\text{-}\wp_{op}(t)) & \text{else} \end{cases} \quad (2)$$

where R denotes the set containing all replicas and $A_{op}(r)$ denotes the set of *all* sets of replicas which satisfy the consistency strategy defined for replica r and operation *op*.

In order to illustrate the formulas given above, we compute the write availability $\wp_w(\mathrm{R}_2)$ of the logical replica R_2 from our example.

$$\begin{aligned}
\wp_w(\mathrm{R}_2) &= \wp_w(\mathrm{R}_6) \cdot \wp_w(\mathrm{R}_7) \cdot \wp_w(\mathrm{R}_8) + \\
&\quad (1 - \wp_w(\mathrm{R}_6)) \cdot \wp_w(\mathrm{R}_7) \cdot \wp_w(\mathrm{R}_8) + \\
&\quad \wp_w(\mathrm{R}_6) \cdot (1 - \wp_w(\mathrm{R}_7)) \cdot \wp_w(\mathrm{R}_8) + \\
&\quad \wp_w(\mathrm{R}_6) \cdot \wp_w(\mathrm{R}_7) \cdot (1 - \wp_w(\mathrm{R}_8)) \\
&= p^3 + 3p^2 \cdot (1 - p) = 3p^2 - 2p^3
\end{aligned}$$

4 An Advanced Algorithm for Embedding Replication Protocols

In order to guarantee the correctness of an embedding strategy it is sufficient that every logical replica uses a combination of consistency strategies which guarantees one-copy serializability. For providing correctness in this sense the read/write consistency, strategy pairs of every logical replica have to be chosen in such a way that there is a non-empty intersection between the sets of members that are required for the execution of each operation. Additionally, any two sets of members which satisfy the write consistency strategy have to intersect. Examples for correct strategy pairs are `one/all` for set structures and `c-cover/c-cover&column` for grid structures. A proof of this statement is straightforward and is not given here.

However, for a special class of embedding replication protocols the correctness condition can be weakened without sacrificing one-copy serializability. The locking for a write operation of a logical grid replica normally consists of two phases. During the first phase a c-cover and during the second phase a full column of members have to be locked with write locks. Following the algorithm described above, every logical grid replica has to execute both phases in order to be successfully locked. We call this algorithm the *standard* embedding algorithm.

Figure 4 shows a two-level grid protocol. The root replica is a logical grid replica as well as all of its members. The replicas marked with "L" are locked by an example `lock(write)` operation following the standard embedding algorithm. Nine real replicas out of 16 have to be locked.

Any two sets of real replicas locked by a correct write operation, intersect in at least four real replicas, although only an intersection in a single replica is necessary. In order to avoid this, we allow all logical grid replicas which are members of another logical grid replica to use a simpler algorithm with only a single phase. These logical grid replicas – when locked during the first phase – need to lock their members only with the first phase algorithm. Analogously, when they are locked during the second phase, locking their members with the second phase algorithm only, is sufficient. Since the set of replicas locked by the first phase and the set of replicas locked by the second phase algorithm have exactly one replica in common, a single replica is locked in both phases which satisfies the consistency requirements. We call this algorithm the *advanced* embedding algorithm. This technique is also used by the Hierarchical Grid Protocol [11]. Figure 5 shows the situation after an example `lock(write)` execution using the advanced embedding algorithm. Only seven real replicas have to be locked by the operation.

Compared with the standard algorithm the advanced algorithm offers reduced costs of a write operation and an increased availability.

A similar improvement can be made for logical set replicas which are members of a logical grid replica. Again, logical replicas which are locked during the column phase of a write operation have to lock their members by using their defined write consistency strategy. But if a logical set replica is locked during the c-cover phase of a write operation, it needs to lock only a sufficient number of its members to satisfy the read consistency strategy. This is an advantage, since usually this number is smaller than the number which is required for write operations. However, for consistency reasons write locks must be acquired to lock the members.

If we use the standard embedding algorithm for our example from section 2.1 in figure 1, locks e.g. on the real replicas R_6, R_7, R_{10}, R_{12}, R_{13}, R_{14}, and R_{15} are required. If we apply the

advanced embedding algorithm, a write operation needs e.g locks on the logical replicas R_2 and R_4 within the c-cover phase and on R_2 and R_3 within the column phase. Since R_4 is locked by using its read consistency strategy, only one of its members has to be locked. This results in write locks e.g. on the real replicas R_6, R_7, R_{10}, R_{12}, and R_{13}. Figure 6 shows the resulting locks on embedding replicated object. The real replicas marked with a bold face "**L**" are the replicas which are locked by both algorithms. The two real replicas marked with the tiny "L" have to be locked by the standard algorithm only.

Following the formula given in equation (1) the costs of a write operation when using the advanced algorithm then computes to

$$\begin{aligned} c_w(R_1) &= \tfrac{1}{4}((c_r(R_2) + c_w(R_4) + c_w(R_5)) + \\ &\quad (c_{\text{c-cover}}(R_3) + c_w(R_4) + c_w(R_5)) + \\ &\quad (c_w(R_2) + c_{\text{column}}(R_3) + c_r(R_4)) + \\ &\quad (c_w(R_2) + c_w(R_3) + c_r(R_5))) \\ &= \tfrac{1}{4}((2+3+4) + (2+3+4) + \\ &\quad (2+2+1) + (2+3+2)) = \mathbf{7.5} \end{aligned}$$

Compared with the standard algorithm which has write costs of 9 the advanced algorithm has lower write costs. As already mentioned it also results in a higher write availability.

5 Conclusions

In this paper, we have proposed the model for embedding replication protocols. These protocols can easily be tailored to fit for a given computer network topology and to a given application. With the help of logical replicas as a building block to set up a hierarchy of replication protocols, it is possible to group the replicas in a such way that minimal communication costs

are obtained and that specific characteristics of the subnetworks can be exploited, e.g. multicast capabilities. We have shown how to analyze embedding replication protocols in terms of costs and availabilities. Additionally, we have presented conditions which are sufficient for the correctness of an embedding strategy and we have shown that for embedding protocols with logical grid replicas some of these constraints can be weakened. This results in the advanced algorithm.

We are currently implementing the model of embedding replication protocols for building a simulator tool. This tool will allow us to map different protocols to specific network topologies and to observe their characteristics under more realistic conditions. Our goal is to make out a catalogue of guidelines on how to construct an embedding replication protocol which suits to a given network topology and provides desired communication costs and availabilities.

References

[1] D. Agrawal and A. El Abbadi. Exploiting Logical Structures in Replicated Databases. *Information Processing Letters*, 33(1):255–260, 1990.

[2] D. Agrawal and A. El Abbadi. The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. In *Proc. of the 16th VLDB Conference*, pages 243–254, 1990.

[3] P. A. Bernstein and N. Goodman. An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases. *ACM Transactions on Distributed Systems*, 9(4), 1984.

[4] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery.* Addison Wesley, 1985. ISBN 0-201-10715-5.

[5] S. Y. Cheung, M. Ahamad, and M. H. Ammar. The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data. In *Proc. of the 6th International Conference on Data Engineering*, pages 438–445, February 1990.

[6] B. Freisleben, H.-H. Koch, and O. Theel. Designing Multi-Level Quorum Schemes for Highly Replicated Data (Extended version). *IEICE Transactions on Information and Systems*, E75-D(6):763–770, 1992.

[7] D. K. Gifford. Weighted Voting for Replicated Data. In *Proc. of the 7th ACM Symposium on Operating Systems Principles*, pages 150–162, 1979.

[8] S. Jajodia and D. Mutchler. Dynamic Voting. In *Proc. of the ACM SIGMOD*, pages 227–238, 1987.

[9] H.-H. Koch. Exploiting Logical Tree Structures for Data Replication Schemes. In *Proc. of the 23rd International Conference of Fault Tolerant Computing Systems*, pages 382–391, June 1993.

[10] A. Kumar. Performance Analysis of a Hierarchical Quorum Consensus Algorithm for Replicated Objects. In *10th International Conference on Distributed Computer Systems*, pages 378–385. IEEE, 1990.

[11] A. Kumar and S. Y. Cheung. A High Available $\sqrt{N}$ Hierarchical Grid Algorithm for Replicated Data. *Information Processing Letters*, 40(0):311–316, 1991.

[12] J.-F. Pâris. Voting with Bystanders. In *9th International Conference on Distributed Computer Systems*, pages 394–401. IEEE, 1989.

[13] J.-F. Pâris and D.D.E. Long. The Performance of Available Copy Protocols for the Management of Replicated Data. *Performance Evaluation*, 11(1):9–30, 1990.

[14] R. H. Thomas. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. *ACM Transactions on Database Systems*, 4(2):180–207, 1979.

List of Figures

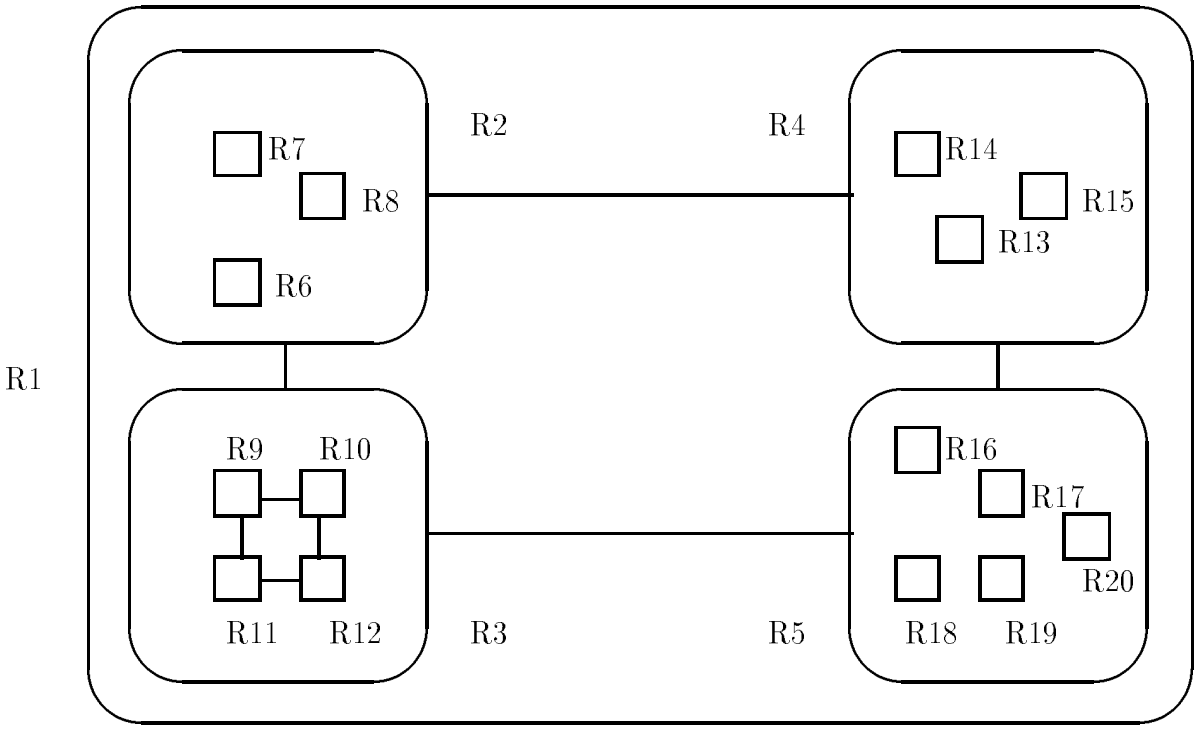

Figure 1: An embedding replication protocol with five logical replicas

Logical Replica	Structure	Read Strategy	Write Strategy
R_1	Grid	`c-cover`	`column&c-cover`
R_2	Set	`majority`	`majority`
R_3	Grid	`c-cover`	`column&c-cover`
R_4	Set	`one`	`all`
R_5	Set	`quorum(2)`	`quorum(4)`

Figure 2: Definition of the logical replicas

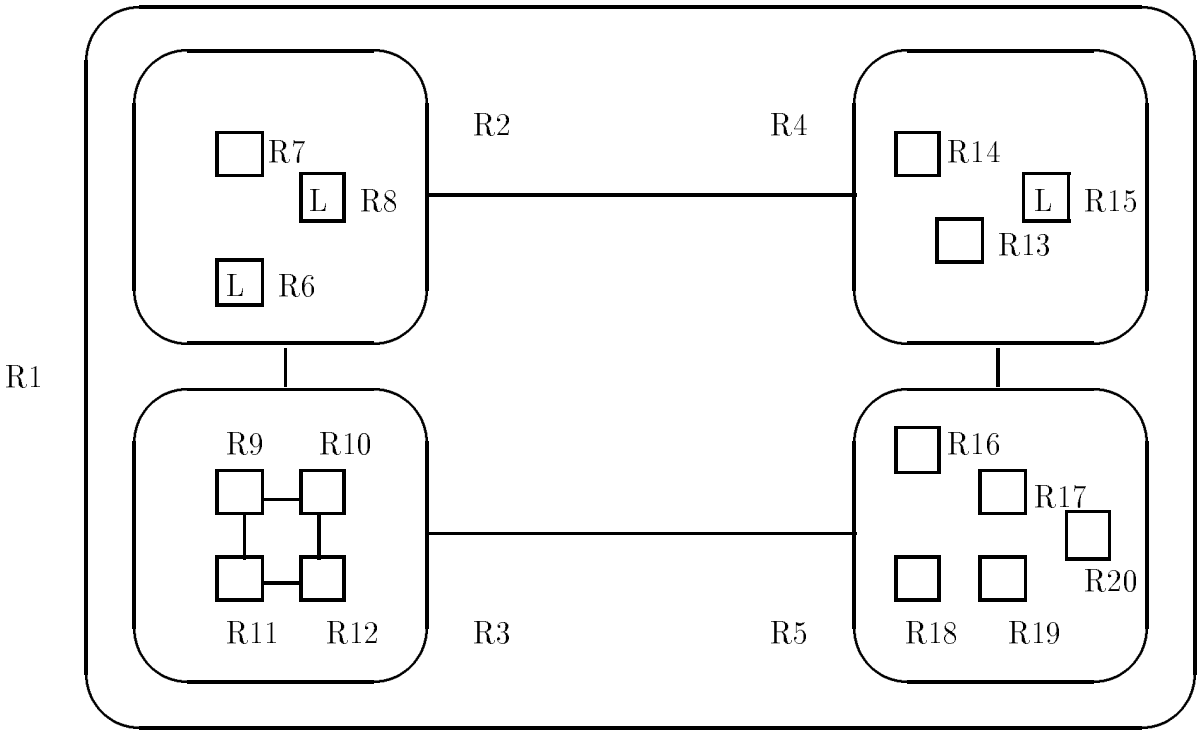

Figure 3: Locks collected by an example read operation

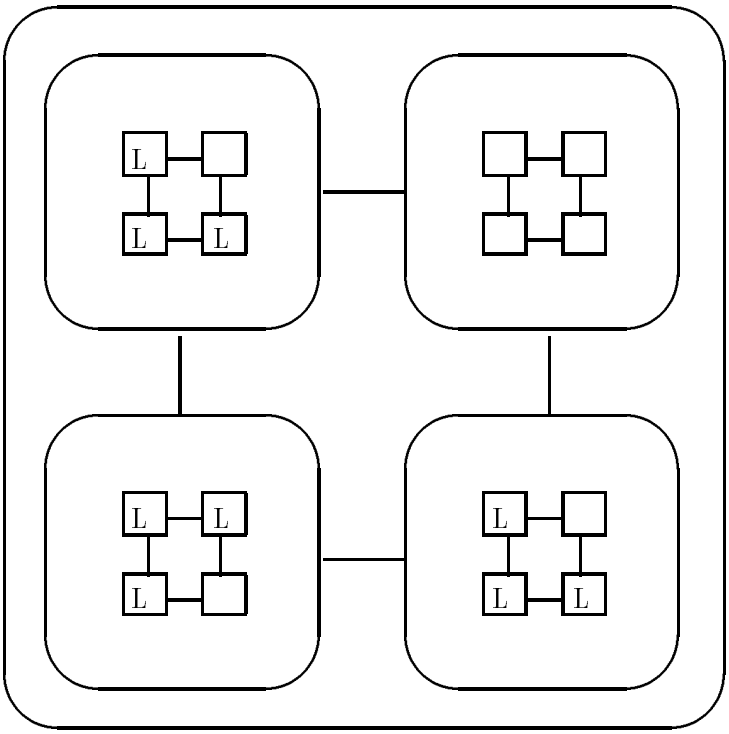

Figure 4: Write locks acquired on a logical grid replica following the standard embedding algorithm

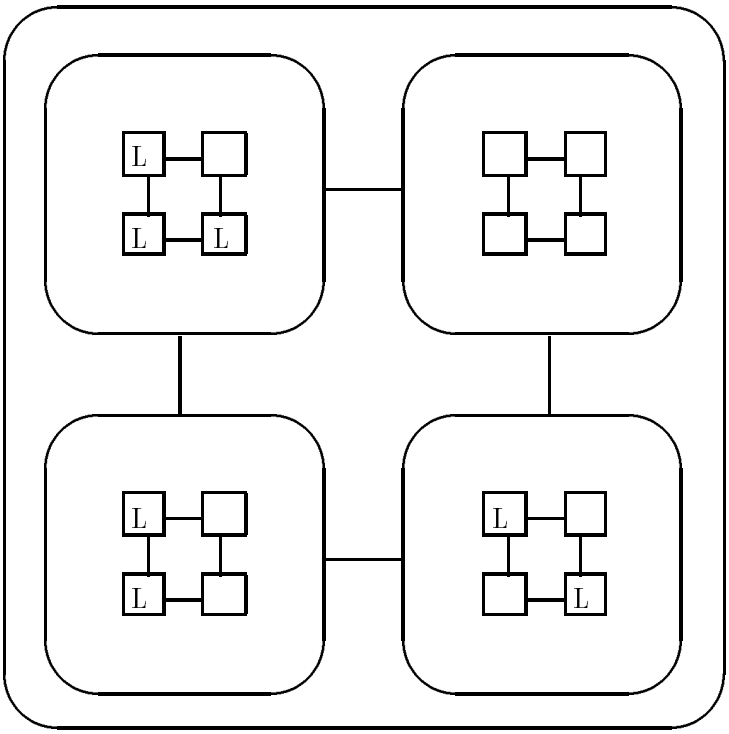

Figure 5: Write locks acquired on a logical grid replica following the advanced embedding algorithm

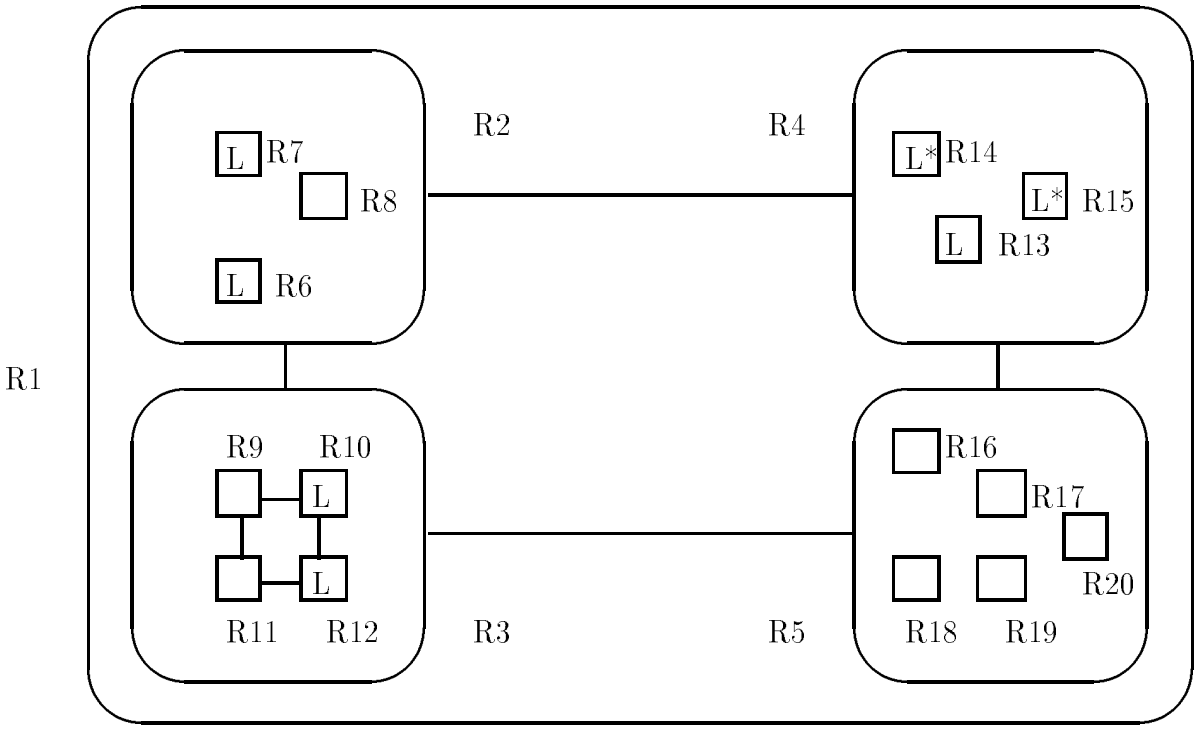

Figure 6: Write locks acquired on the example object following the standard and the advanced embedding algorithm