Decentralized Managing of Replication Objects in Massively Distributed Systems

Daniel Klan, Kai-Uwe Sattler, Katja Hose, Marcel Karnstedt
Department of Computer Science & Automation
TU Ilmenau, Germany
{daniel.klan,kus,katja.hose,marcel.karnstedt}@tu-ilmenau.de

ABSTRACT

Data replication is a central technique to increase availability and performance of distributed systems. While offering many advantages it also requires more effort for ensuring data consistency in case of updates. In the research literature various approaches for replication management in distributed databases have been presented, but they are mostly limited either in scalability or in the consistency guarantees they provide. On the other hand, P2P systems usually provide replication support but ignore the update problem.

In this paper we present a new approach for managing replicated data in wide area distributed networks. Our solution is orthogonal to the underlying infrastructure and managed in a decentralized manner. It guarantees single-master consistency and allows updates at any node of the system by combining traditional replication techniques with ideas known from P2P systems.

1. INTRODUCTION

Being initiated by the popularity of file sharing applications P2P systems gain more and more attention. Decentralized P2P systems like Gnutella or FreeNet offer an alternative to traditional client-server architectures. Decentralization P2P systems allow a fair distribution of load and avoid a single point of failure. Therefore, they usually scale well in very large environments like the Internet.

Wikipedia is one of the most popular online encyclopedia projects in the World Wide Web. The project contains more than 9 million articles distributed on more than hundred serves [1]. Wikipedia is a non-profit-project, financed by donations. Maintaining and supporting such a large project is associated with high costs. In spite of the complex architecture and the high number of servers the configuration has also single point of failures, the master database server. Using a decentralized massively distributed system, where each user stores a part of the articles, would decrease costs and increase availability. Each user of the data management system must be able to update each data item, independent of the location of this item. Once changed, an update must not lost.

P2P systems like UniStore [17] offer techniques to distribute and query data like articles. In general, P2P system do not support updates. The main reason is the original application area - file sharing. On file sharing systems, updates are not necessary. Changed versions are inserted as new files. Additional, in case the interest in a file decreases, the file is extincted.

A key concept of P2P systems for improving availability of data objects and enabling efficient parallel processing of queries is data replication. However, due to the absence of a central instance several problems arise. Usually, structured overlays like Chord [25], CAN [20], and P-Grid [2] provide some basic support for replication. They allow to keep copies of data objects which can be retrieved during a lookup but typically lack support for maintaining replicas. Even though some P2P systems support updates of data, the updates are usually restricted to the node storing the primary copy, meaning that they behave like traditional primary copy resp. single master systems. Thus, they share all the well known drawbacks like having a single point of failure and a high workload for the master. In case the node responsible for the primary copy fails, no further updates of this data object are possible.

Another problem related to replication mechanisms is the localization of data copies. While maintenance of data copies in small environments with a small number copies is easy, it is very hard in large-scale environments with unreliable nodes and often hundreds or thousands of replicas [9]. Some P2P systems use their logical network structure to propagate and locate data copies. For example on Chord [25] replicas are propagated around the ring structure. Other systems propagate data copies to a randomly chosen set of nodes. The structured overlay network P-Grid uses randomized replica distribution also for load balancing [4, 5]. Hence, maintenance of data copy locations and consequently data updates are in general very expensive in systems for a larger number of replicas.

In summary, we identify the following challenges for replication in P2P systems:

- robustness, i.e. supporting updates even if the primary owner of a given data object fails without risking consistency,
- scalability wrt. the number of replicas in order to support settings with hundreds or thousands of replicas.

In this paper we present BORG (B*org* - *g*O*verning* R*epli-*

DAMAP'08 , March 25, 2008, Nantes, France.

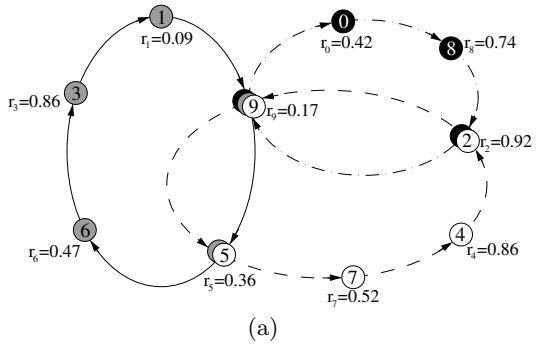
*cation objects in lar*G*e environments*), a new approach for maintaining data copies in P2P systems addressing these challenges. The maintenance process is completely decentralized and orthogonal to the underlying P2P system. In BORG any node storing a data copy is able to initiate updates, while at the same time single master consistency, well known from central systems, is guaranteed. In a second step, we extend our system to improve the update process so that it allows to include some node characteristics in the replication strategy.

The remainder of this paper is structured as follows. After a brief survey on related work in Section 2 we introduce our approach on data copy management in Section 3. Then, we present our method for integrating node characteristics into the replication process in Section 4. Finally, we report results from our evaluation. We proceed by presenting simulation results referring to stability and scalability of our replication framework (Section 5) and finish with a conclusion in Section 6.

2. RELATED WORK

Algorithms for maintaining replicated data can roughly be divided into two classes. On the one hand there are optimistic methods, which are based on the "optimistic" assumption that problems occur rarely [18, 22]. Conflicts are fixed after they have happened. The flexibility and high availability of optimistic systems is typically at the expense of weaker consistency guarantees. Main tasks on any optimistic replication system are conflict detection and conflict resolution. In case a reconciliation (for instance by semantic transformation) is not possible, rollbacks are necessary. In [14] Gray et. al showed that optimistic approaches are not suitable for very large systems since the likelihood of collisions is quadratic in the number of nodes with write access. Typical systems using optimistic replication strategies are the domain name service [7] and the Usenet [24]. Updates in this systems are very rarely or do not occur.

On the other hand there are traditional pessimistic replication algorithms which give users the impression of working with a single copy [8]. Commonly used in commercial systems are *primary copy systems*, where each data object is managed by its own master node [22]. Thus, these systems have the same disadvantages that classical client-server architectures have to deal with. To overcome the difficulties of centralized architectures different decentralized controlling mechanisms have been developed. Popular pessimistic decentralized mechanism are voting procedures. The basic idea of voting procedures is that an operation requires a predefined quorum of votes to perform a restricted operation. A single node may have a number of votes assigned to it. Serialization of updates is attained that any two potential quorums must share at least one replica. In literature various voting strategies are presented, like majority consensus [26], dynamic voting [19, 16] or different weighted voting strategies [13, 6]. Although voting-algorithms are decentralized, they usually require a central instance for managing information about the data object or metadata. To overcome these disadvantages Rodrig and LaMarca introduced a weighted-voting algorithm [21] that distributes the metadata along with the data. In order to update a data object a number of replication objects sufficing the predefined quorum have to be locked. To ensure locking of the involved nodes a message has to be sent to all nodes in the system every time a data object is changed.



(a)

Figure 1: BORG network with three replication rings, the nodes 2 and 5 store two replicas each, node 9 stores a copy of all three replication objects

To illustrate that aspect suppose that we desire a data object availability of 0.999 while having an average node availability of 0.04, that is a node is available one hour per day. This assumption is not unrealistic. Chen et. all show in [11] that more than 30 percent of nodes within a P2P network have session length of 10 minutes or less. In the example scenario more than 160 data copies are necessary [9]. Using a voting algorithm for any update 160 messages as well as the corresponding answers are necessary! In contrast in BORG only a few messages are necessary. In case of repeated updates without changes on the network structure O(1) messages are sufficient. Additional Bhagwan et. all show in [9] the necessity to refresh the distribution of copies in a P2P system. Otherwise, data will completely lost due to the decay caused by the leave of nodes.

3. THE BORG SYSTEM

BORG is a decentralized organized overlay network for maintenance data copies in wide-area distributed systems. The system is orthogonal to the underlying network, i.e., no further assumptions on the physical network structure is required. Our system is a pointer structure, not a DHT, that is we do not consider key lookups within the network. This is a typical task of the underlying system, e.g. appropriate routing mechanisms like DHT's [3, 25, 20]. The following section gives an introduction to BORG, explaining its basic principles, the underlying routing structure and aspects of system maintenance.

3.1 Architecture

Due to the manifold usage of P2P systems different types of replication objects may be relevant. In systems like Gnutella and Bittorrent only files are shared, that is, potential replication objects may be files, sets of files or all files managed by a single node. In opposition consider P2P systems with database functionality, where replication objects like tuples, tables or tablespaces are suitable. To ease the subsequent description of our system, we consider a node's entire data as a single replication object, independent of the node's type. Regarding the distributed Wikipedia example from the introduction, we can consider an article or a collection of articles as replication object.

Within BORG all nodes owning a copy of the same repli-

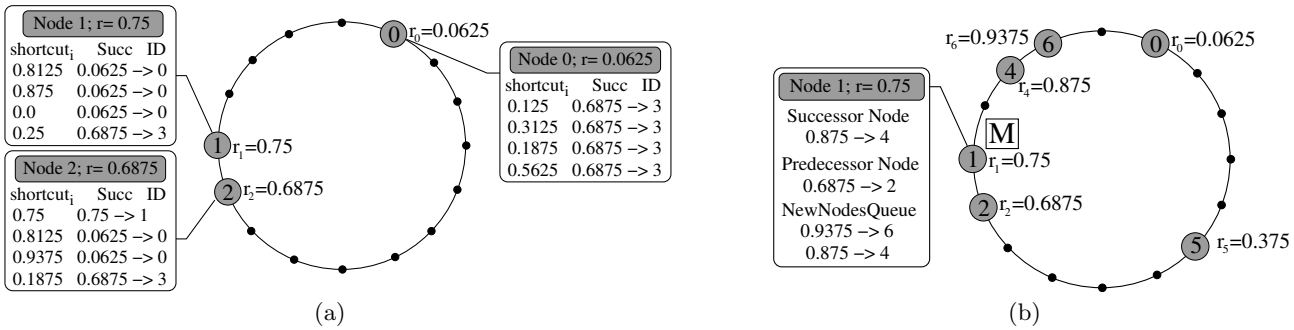


Figure 2: (a) a replication ring with 3 nodes; (b) Replication during an update process, the nodes 4, 5 and 6 joined the replication ring during the running update; M denotes the current temporary master

Algorithm 1 Basic update algorithm initiated by node n

```
1: update n's local data
2: initiate master lookup
3: propagate update to the temporary master
   (using operation transfer)
4: if (the update conflicts with a running update on the master) and
   (a conflict resolution is not possible) then
5:    reject update
6:    notify node n for local rollback
7: else
8:    propagate the changes to all nodes in the replication ring
      (using state transfer and, for instance, a counterclockwise prop-
      agation strategy)
9:    notify n about the successful update
10: end if
```

cation object are organized in a virtual ring, denoted as *replication ring*. Each node holding a copy of a given replication object has knowledge of only one node, its successor within the corresponding replication ring. All rings in BORG are independent and managed in a completely decentralized manner. Each node may hold copies of several replication objects. Thus, each node might be part of several replication rings – one for each of the replication objects. BORG has been designed for managing partial replication in widely distributed P2P systems, where each node stores only a small portion of the global data. However, it can also be used for decentralized full replication systems. In that case there would only be one replication ring containing all existing nodes.

The nodes within a ring are sorted by a key r. This could for instance be a randomly chosen value or, as presented in Section 4, a measure reflecting knowledge about the characteristics of a node. For simplification we choose $r \in [0, 1]$. Since the keys are used to sort nodes within the rings (i.e., the nodes are ranked), we refer to r as a *ranking measure*. Figure 1(a) shows a sample BORG system with 10 nodes, 3 replication objects and their corresponding replication rings.

3.2 Update Process

In order to update a data record a node has to process the steps illustrated in Algorithm 1. At first, changes are applied to the node's local copy and then a procedure is initiated to identify a node as *temporary master* within the corresponding replication ring.

In order to guarantee that all nodes within a replication ring identify the same temporary master, the node of the ring with the highest rank r is chosen as master. To prevent two nodes from being the master at the same time, we do not permit equally ranked nodes within a replication ring. Thus, the identification process is reduced to a simple lookup of the unique node with the highest rank r within the ring. This node is unambiguously characterized by being the only node within the ring having a successor node with a smaller rank than its own.

The node determined as master in the course of an update is responsible for managing this single update, i.e., it checks for possible conflicts, manages them (for instance by blocking or aborting operations), and if necessary propagates successful changes to the remaining nodes within the ring. In order to detect a node failure during the update process, the temporary master sends a success message to the update initiator. In case the initiator does not receive a success messages the update will be initiated again.

If a conflict resolution is not possible, the master notifies the update initiator which undo its local changes. Successful updates are then propagated counterclockwise using an additional predecessor link. With this, we guarantee that future master nodes are up-to-date. For the counterclockwise propagation $O(N)$ messages are necessary.

3.3 Routing and Reliability

Since nodes only know their successor within a ring, a lookup message has to be forwarded around the ring. Consequently, a lookup could be very expensive. Additionally, utilizing the successor list for routing messages is not reliable, since just one failing node between the lookup initiator and the target node causes the whole lookup to fail. In order to improve the lookup performance and to increase reliability we introduce shortcuts within the replication ring in accordance to the fingers utilized by Chord [25].

For the computation of the routing table we restrict the maximum number of nodes that may join a replication ring to N. Then, the i-th shortcut, $0 \leq i \leq \log(N) - 1$, of node n points to the node with the smallest rank r equal to or higher than

$$shortcut_n^i = \frac{(r_n \cdot N + 2^i) \bmod N}{N} \tag{1}$$

where r_n denotes the rank of node n itself. The first shortcut of node n is a link to n's successor within the ring. Figure 2(a) depicts a replication ring with 3 nodes and their routing tables. Since N determines the granularity of possible ranks r in a replication ring, N should be chosen sufficiently large

for a given application scenario.

Using the logarithmic routing table the distance between two nodes within the same replication ring is at least halved. Thus, the identification of the master node requires $O(\log N)$ messages. Additionally, the reliability of the replication ring structure is increased since an unavailable node can be bridged by forwarding the message to a closer node known from the routing table.

3.4 Ring Maintenance

Once a new replication object is created a new replication ring for managing this new object has to be set up. Subsequently, the ring assimilates nodes up to a predefined number of k nodes, where $k \ll N$. To assure that k copies of an replication object exist, failed nodes are replaced by new ones. For assessing the number of nodes within a replication ring we may either apply a token technique or the method proposed in [10]. The latter estimate the number of nodes within a ring using the knowledge of the shortcut tables.

In BORG the maximum number k of nodes within a ring can either be a predefined value or a repeatedly computed dynamic value that depends on the required availability of the replication object related to this ring. In case k is a dynamic value replication rings may contain different numbers of nodes.

Adding a Node.

In case the number of members within a replication ring is too small one of the ring's nodes, the temporary master, initiates a lookup for new members by contacting nodes known from the routing table of the underlying network. If a candidate node is identified its suitability is checked. A node is appropriate for joining the replication ring (i) if it is not already a participant of the ring, (ii) if it has a rank which is different from that of the current ring members, and (iii) if it is not a member of too many other replication rings. The latter prevents overloaded nodes.

If a node is suitable for ring membership, it checks if it conflicts with an existing temporary master of the ring. This is the case when an update is currently processed and the new node's rank is higher than that of a potential temporary master. For this purpose, a node compares its own rank to the one of its successor and checks whether there is any node marked as master. Only if both conditions are true, the new node is in conflict and needs special treatment. Otherwise, the new node can be integrated utilizing the *join* method known from [25].

When a conflict has been identified, this means that an update is currently running and the new node has to notify the current master. The master maintains a queue that contains all nodes having a higher rank than its own, i.e., a list of master candidates that joined the ring during the currently running update process (see Figure 2(b)). Subsequently, the new node stores the identifier of the running master and joins the ring. In case a joined node with a rank higher than that of the running master receives a request for a new update, it forwards the request to the current master. Once the running master has finished all concurrent update processes, all nodes of the queue are notified about the end of the update and get the last changes. It starts with the highest ranked node to provide changes to the most promising master candidates first. To prevent a running master node from being continuously occupied by update processes, while having nodes with a higher rank that joined the ring, it is useful to introduce a time threshold for the runtime of an update. In case of a timeout the update process of the current master is interrupted and changes as well as information on unprocessed updates are forwarded to the node currently holding the highest rank.

Determination of a new node's position within a ring and subsequent conflict checking require $O(\log N)$ messages. For the insertion $O(\log^2 N)$ messages are needed [25].

4. THE RANKING MEASURE

In Section 3 we have discussed how to identify a unique master utilizing the node's ranks. We argued that choosing an arbitrary value as rank, for instance a constant value generated randomly, would be sufficient for the algorithm to work properly. An alternative to using random values as node ranks is to determine a node's rank depending on its specific properties. This allows to influence the replication process in a way that improves both performance and reliability of the update process. In this section, we identify node specific properties that have to be reflected in the ranking measure to achieve these goals.

4.1 Node characteristics

Performance.

BORG has been developed for managing replicas in decentralized environments. However, updates are performed by a (temporary) master node which is usually subject to higher workload than the remaining nodes within a replication ring. Hence, the master should be the most powerful node within a given ring. By integrating knowledge about technical properties like CPU performance, network bandwidth, and data access time into the ranking measure, powerful nodes are ranked higher than powerless nodes within the same ring. Thus the temporary master, which is the node with the highest rank in a ring, is appropriate for performing an update.

In a static non-dynamic system, where nodes do not join or leave a replication ring and a node's rank does not change, updates are repeatedly forwarded to the same master node. To prevent an overload of this node, it is also useful to integrate the node's workload in the ranking measure. A higher load caused by repeated update processes decreases the node's rank and thus makes it less likely to be chosen as master.

Reliability.

In BORG a node has only a limited view on the locations of a replication object's copies. Hence, node failures are more problematic than in systems with global knowledge. For this reason we use a *stability* procedure which corrects broken shortcuts periodically. Nevertheless, as long as a shortcut is broken problems on message forwarding may occur. In order to improve the reachability of the temporary master and master candidates we include the average online availability in our ranking measure. That means, a higher online availability of a node increases its rank.

As argued above several criteria of a system affect the execution time of a query and the system's reliability. In general, we can differ between *static* and *dynamic* properties of a node. Static criteria are technical properties which

generally do not change. That is, in case of static criteria it is sufficient to determine the quality of a node only once, for instance at the application startup. On the other hand, dynamic criteria are usually time dependent and have to be determined continuously. Consequently, the rank has to be adapted permanently and repeated corrections of the node's position within all rings, that it is member of, are necessary.

4.2 Updating a node's ring position

Whenever a node's rank has changed an adaption of its position in all replication rings where it is member of is necessary. In order to reduce the effort for ring maintenance suitable position update strategies are essential. In the following we present appropriate solutions for handling resp. preventing position updates depending on the extent of the changes to the rank r. We distinguish the three situations: (i) low fluctuations of r, (ii) small and (iii) significant changes to r.

Low fluctuations.

A basic approach to reduce the effort for position updates is to change a node's position only if its state has changed reasonably. To prevent changes of r due to low fluctuations of a node's properties it is useful to define the current rank depending of its history. For example we introduce a smooth ranking measure r'. The value of r' at time $(i+1)$ is determined by the measured rank at time $i+1$ and the value of r' at time i. The overall smooth ranking measure can be described by the following recursive definition

$$\begin{aligned} r'(0) &= r(0) \\ r'(i+1) &= \alpha \cdot r'(i) + (1-\alpha) \cdot r(i+1) \end{aligned} \tag{2}$$

where $\alpha \in [0,1)$ represents the strength of the memory. If α is low, old values will be forgotten faster. Otherwise, with higher values of α the influence of old values grows.

Additionally, we introduce a threshold τ to prevent that small changes to a node's characteristics result in changes to its rank and thus a position update. Hence, the position of the node within a ring is only updated, if its current rank is increased resp. decreased by more than τ. Figure 3 shows a small example combining both presented techniques. Relying on the ranking measure r a high number of position updates are necessary. In contrast, using the smooth ranking measure r' and a threshold τ only 3 updates are needed (at time 14, 38 and 93).

Small changes.

In case a node's rank exceed the threshold, adjustments of its position within all rings, that it is member of, are necessary to preserve the order of the nodes.

In BORG there are two options for locating nodes within a replication ring. On the one hand we can do this via the ring structure, i.e., using the successor links. On the other hand we may do this via the shortcut table utilizing the shortcuts used for the lookup process. In order to discover a temporary master a correct successor link is essential, since each node has to check whether its successor has a smaller rank than itself. In contrast, a shortcut table with some incorrect entries only increases the number of messages for the master lookup but will not cause the lookup to fail.

Thus, in case a node's rank is changing only slightly, it is sufficient to let the node slowly "float down" within the ring. This means that we adjust successor links immediately

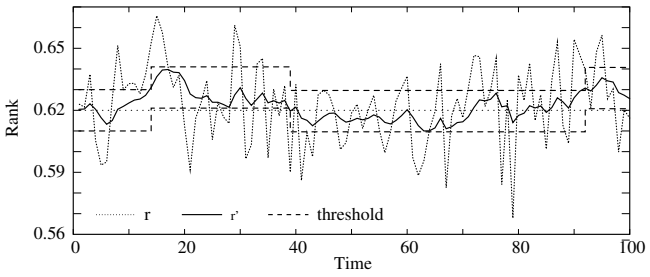


Figure 3: Simulation of a ranking measure r reflecting dynamic node properties, its corresponding smooth ranking measure r' and the threshold τ. We choose $r = 0.62$ with an additional Gaussian noise $N(0, 0.02)$, the memory α is set to 0.15 and $\tau = 0.01$.

and do necessary adaptations of the shortcuts list lazily by the *stability* process that verifies the validness of the single fingers.

W.l.o.g. let us consider a node whose rank is decreasing. For simplicity we assume that this node is a member of just one replication ring and it does not have the smallest rank within this ring. Adjustments of the successor links are done as follows. We distinguish three cases that may occur: the rank of the node is (i) higher, (ii) equal or (iii) lower than the rank of its predecessor.

(i) In case the rank of the node is higher than that of its predecessor, an adaption of the successor link of the predecessor node is necessary.

(ii) If the node's rank is decreased and conflicts with the rank of the node's predecessor, no adaption is done, since two equally ranked nodes within the same replication ring are not allowed. The node itself keeps in mind that its rank is decreased, but externally holds its old rank. Once the rank is decreased again an adaption is initiated. Consequently, during a master lookup, a node may be identified as master although it has the same rank as its predecessor.

(iii) In case the node's new rank is smaller than the rank of its predecessor, the node and its predecessor have to exchange their position within the replication ring. That is, both nodes and the predecessor of the node's predecessor have to adjust their successor links.

Thus, letting a node "float down" only requires messages to be exchanged between the node and its predecessor resp. its two predecessor nodes, $O(1)$ messages are needed for the node's ranking update.

Significant changes.

The two techniques described above are applicable if only small changes to characteristics of dynamic properties occur. Whenever a node's rank changes significantly (for instance due to a hardware upgrade that changes the node's properties considerably), it is not appropriate to let it "float down" Hence, a reinsertion of the node is required, that is its removal from all K replication rings where it is member of and its subsequent reinsertion into these rings. Therefore, $O(K \cdot \log^2 N)$ messages are needed.

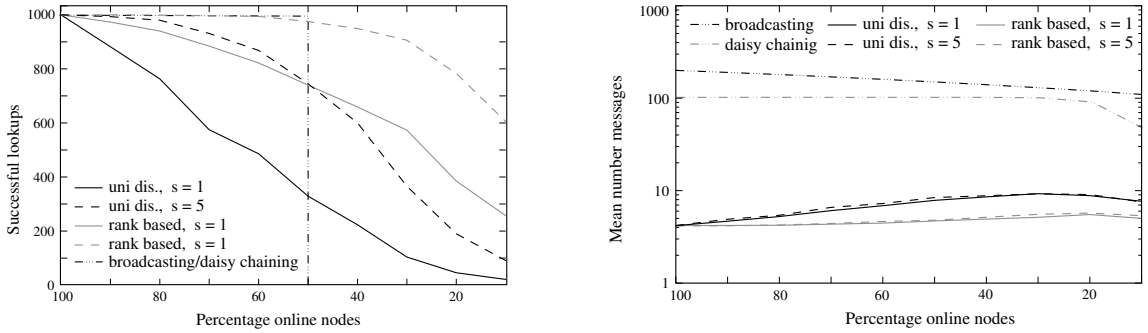


Figure 4: Initiating updates in a replication ring/network with 100 nodes, some nodes are offline: (a) number of successful lookups on BORG (averaged about 1000 runs) and analytical success rate of two majority consensus strategies, (b) the corresponding number of messages

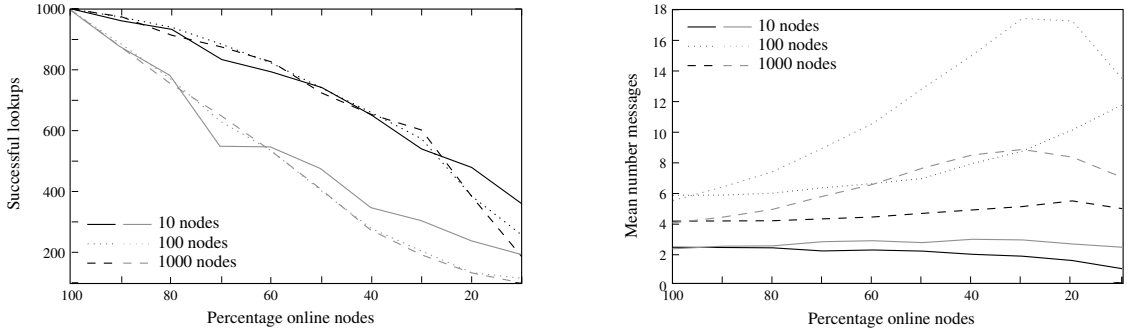


Figure 5: Master lookups in replication rings with 10, 100, and 1000 uniformly distributed (dark lines) nodes and clustered (light line) nodes, some nodes are offline (rank based distribution): (a) number of successful lookups in 1000 runs, (b) the corresponding number of messages, note the logarithmic scale

5. EVALUATION AND DISCUSSION

In our simulative evaluation of BORG we assessed the reliability of its master lookup process as well as scalability and performance aspects. We simulate only one replication ring. The reader should keep in mind that in a real BORG system using a dynamic rank activities in one replication ring may affect those in another. This means that a high workload of a node caused for instance by an update process in one ring affects the node's rank and therefore its position in another ring. We abstract from this aspects, which are subject to further investigations in future research. The nodes' ranks are created by a random generator. For all simulations the maximal number of participants N in a replication ring is set to 2^{20}, that is each routing table contains max. 20 links.

Stability.

In a first experiment we examined the reliability of the master lookup process in case of node failures within a replication ring. Lookup messages that cannot be delivered due to a node failure are forwarded to preceding nodes of the original receiver (within the ring) utilizing the links from the routing table. In case where no such node exists, the lookup finishes unsuccessfully.

We evaluated the reliability of the lookup process in two settings, one simulating a system with a ranking measure that reflects the availability of a node and another without. In the latter setting we simply choose nodes to fail within a ring randomly (*random distribution strategy*), in the first setting we implemented a strategy based on a *rank based distribution* that ensures that nodes with a high rank (due to a high availability) are less likely to fail and thus are rarely chosen to be failed. To realize this strategy we first compute a value $z = r \cdot \xi$ for each node, where r denotes to the node's rank and ξ is a value uniformly distributed in $(0, 1)$. The nodes with the lowest values for z are chosen to be offline.

Additionally, we examined the influence of an successor list, containing a sequence of the first s successor nodes of a peer n. Utilizing this list we are able to detect whether a node is the temporary master or not even in case a number of successor peers failed.

In order to have representative results all experiments are repeated 1000 times. For each pass the nodes' ranks, its routing tables as well as the nodes chosen to be offline and the initiator for the master lookup are rebuild. For comparison, we add the analytical success rate of the two majority consensus strategies broadcast and daisy chaining [26]. We assume that more than 50 percent of the nodes have to ac-

cept a write operation to be performed. Concerning the broadcast strategy we premised that each node knows all other nodes within the replication system, for daisy chaining we assumed that each node knows its 20 successors within the chain.

Figure 4 (a) shows the number of successful lookups for a replication ring with 100 nodes. Please notice that the figure shows just a snapshot, that is we examined cases where 10, 20, ... percent of the nodes failed at the same time. In our final system implementation there will be a stability function available that repairs broken shortcuts. Nevertheless, the results show, that updates in BORG can be initiated (by successfully identifying a temporary master) even though a high number of node failed. In contrast using quorum based strategies, updates are not possible in case of more than 50 percent of the nodes are offline. As expected a combination of the rank based distribution and a successor list is the most successful BORG strategy. The results indicate, that this is not solely due to the usage of a successor list. Utilizing a performance based ranking measure increases the reliability of the lookup process as well as the usage of additional information like the successor list does. As expected performance based ranks master candidates, i.e., nodes with high ranks are more reliable than nodes with low ranks.

Figure 4 (b) illustrates the average number of messages needed for the master lookup (the ordinate is scaled logarithmic). The figure shows the advantages of the BORG strategy compared to the majority consensus strategies broadcast and daisy chaining. Using the shortcuts merely a small number of messages is necessary to initiate an update. In contrast to voting strategies, all node are requested.

Scalability.

In addition reliability aspects of BORG, we have examined its scalability. For this purpose, we simulated replication rings containing 10, 100 and 1000 nodes, without a successor list. Offline nodes were chosen by the rank based strategy, described above.

Figure 5(a) (dark lines) indicates that the lookup success rate is relatively independent from the number of participating nodes. Figure 5(b) (dark lines) shows the required number of messages. As anticipated the more nodes we have in a replication ring the more messages are needed. Increasing the number of offline nodes in a replication ring by 10 participants reduces the number of messages to be sent. In this case, a node is able to find a current master solely by relying on its routing table. In case of 1000 participants the number of messages increases with the number of offline nodes. Due to the high number of participants, it is likely to identify an online node using the routing table. In order to identify this one, all nodes closer to the target have to be tested by an additional message.

Maintenance.

We have also examined the number of messages in case a node joins a replication ring. Figure 6 shows the number of message needed for inserting a node into replication rings with up to 500 participants. Nodes within the replication rings are distributed uniformly. For comparison, we additionally plotted the number of messages needed for inserting a node into a replication system using the both majority consensus strategies. The figure indicates that adding a node using replication rings is more expensive than a simple daisy chaining strategy. Nevertheless, only one update is sufficient to make our replication strategy more efficient than daisy chaining. We do not have to take node failures, because nodes test and repair broken shortcuts continuously.

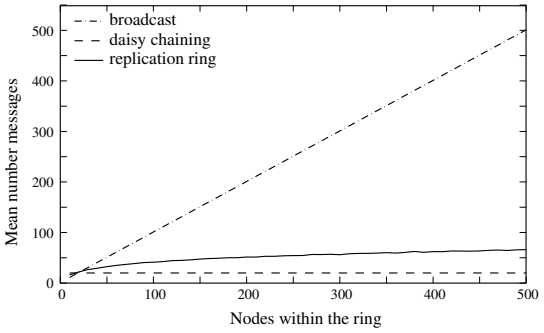


Figure 6: Number of messages for node joining

Node clustering.

Finally we investigated the influence of stability on the node distribution within a replication ring. For this purpose, we compared a setting where the nodes' ranks were distributed *uniformly* and another where the nodes' ranks were *clustered* over the interval of possible ranks. The first setting is exemplary for applications that are made of nodes with very different system properties and therefore a wide range of different ranks. A typical scenario is the Internet. In the second setting nodes' ranks are normally distributed over the interval $[0.95, 1]$. A typical use case for this simulation are cluster systems which usually consist of a number of nodes that have the same hardware configuration. Consequently, the ranks of the nodes are almost equal. They just differ because of the influence of dynamic properties.

We again simulate replication rings with 10, 100, and 1000 nodes using a rank based distribution. Figure 5(a) illustrates that the master lookup on a ring with clustered nodes (light line) is less successful than on a ring with uniformly distributed nodes (dark lines). An explanation is, that in clustered distribution a large number of shortcuts refers to the same successor node. If it fails all shortcuts referencing this node are broken. Figure 5(b) shows the corresponding number of messages. As we can see in a clustered setting the number of messages increases with the number of offline nodes. Since all nodes have nearly the same rank the advantage of the performance based rank is lost.

Increasing the number of ring members is a simple solution to improve reliability in replication rings with clusters. Nevertheless, uniform distribution is a better solution. In case the participants of a BORG systems are known, we also can map the clustered ranks to the ranking space $(0, 1)$.

In Summary, the simulation results show that BORG is suitable to identify a master even when a number of nodes fails. We compare different setups and showed that replication rings are effective with randomly distributed nodes. Additionally, we have shown the advantage of integrating node availability into the replication process. The tests also show that the system cannot compensate a high number of unavailable nodes. Hence, it is necessary to correct broken shortcuts continuously.

6. CONCLUSION AND FUTURE WORK

Replication is one of the main techniques to improve availability and performance in distributed systems. In the literature many approaches are proposed to manage distributed copies. However, classic solutions do not scale well for large systems.

In this paper we have introduced BORG a novel technique for managing massively distributed replicas in P2P systems. Our system is orthogonal to the used P2P application and managed decentralized. Furthermore, it scales well with the number of copies and nodes within the system. We have also introduced a ranking technique that considers node characteristics in the replication process. By means of a simulation we have shown the applicability of our replication system and demonstrated the advantages using a rank measure based on node characteristics.

At current state, all experiments reported in this paper were performed using a simulation environment. As next step, we plan to incorporate BORG into a real DHT-based overlay network for replication management.

One additional aspect of our future work is to work on data reliability. Due to the decentralized architecture of BORG there is no global knowledge. However, we can use the structure to estimate the number of nodes participating in a ring and then compute the number of necessary nodes for a desired data object availability. Furthermore, we have considered in this paper only the simple counterclockwise propagation strategy. In future work, we plan to investigate further propagation strategies like ring flooding or epidemic propagation [12].

Another aspect of future work is to allow a controlled relaxation of consistency [15, 23]. Since updates are always propagated counterclockwise, it is possible to use this knowledge for a consistency relaxation. During a lookup for the current master, for instance in order to get the latest version of a data item, the data on the lookup path become more up-to-date as they are nearer to the master.

7. REFERENCES

[1] http://en.wikipedia.org/wiki/Wikipedia.

[2] Aberer, K. A Self-Organizing Access Structure for P2P Information Systems. In *CoopIS* (2001).

[3] Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R., and Wu, J. Advanced peer-to-peer networking: The P-Grid System and its Applications. *PIK Journal* (2003).

[4] Aberer, K., Datta, A., and Hauswirth, M. Multifaceted simultaneous load balancing in dht-based p2p systems: A new game with old balls and bins. In *Self-star Properties in Complex Information Systems* (2005).

[5] Aberer, K., Datta, A., Hauswirth, M., and Schmidt, R. Indexing data-oriented overlay networks. In *VLDB* (2005), pp. 685–696.

[6] Agrawal, D., and Abbadi, A. E. The tree quorum protocol: an efficient approach for managing replicated data. In *VLDB* (1990), pp. 243–254.

[7] Albitz, P., and Liu, C. *DNS and BIND*. O'Reilly, 2006.

[8] Bernstein, P., Hadzilacos, V., and Goodman, N. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[9] Bhagwan, R., Moore, D., Savage, S., and Voelker, G. Replication strategies for highly available peer-to-peer storage. In *Fudico* (2003).

[10] Binzenhöfer, A., Staehle, D., and Henjes, R. Estimating the size of a chord ring. Tech. Rep. 348, University of Würzburg, 2004.

[11] Chu, J., Labonte, K., and Levine, B. Availability and locality measurements of peer-to-peer file systems. In *TCom: Scalability and Traffic Control in IP Networks II* (2002).

[12] Eugster, P., Guerraoui, R., Kermarrec, A.-M., and Massoulié, L. From epidemics to distributed computing. *IEEE Computer 37*, 5 (2004), 60–67.

[13] Gifford, D. K. Weighted voting for replicated data. In *SOSP* (1979), pp. 150–162.

[14] Gray, J., Helland, P., O'Neil, P., and Shasha, D. The dangers of replication and a solution. In *ACM SIGMOD* (1996), pp. 173–182.

[15] Guo, H., Larson, P.-A., Ramakrishnan, R., and Goldstein, J. Relaxed currency and consistency: how to say "good enough" in sql. In *ACM SIGMOD* (2004), pp. 815–826.

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

[17] Karnstedt, M., Sattler, K., Richtarsky, M., Müller, J., Hauswirth, M., Schmidt, R., and John, R. UniStore: Querying a DHT-based Universal Storage. In *ICDE, Demonstrations Program* (2007), pp. 1503–1504.

[18] Page, T. W., Jr., Guy, R. G., Heidemann, J. S., Ratner, D. H., Reiher, P. L., Goel, A., Kuenning, G. H., and Popek, G. Perspectives on optimistically replicated peer-to-peer filing. *Software – Practice and Experience 11*, 1 (1997).

[19] Pâris, J.-F., and Long, D. D. E. Efficient dynamic voting algorithms. In *ICDE* (1988), pp. 268–275.

[20] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. A Scalable Content Addressable Network. In *SIGCOMM* (2001), pp. 161–172.

[21] Rodrig, M., and LaMarca, A. Decentralized weighted voting for p2p data management. In *MobiDE* (2003), pp. 85–92.

[22] Saito, Y., and Shapiro, M. Optimistic replication. *Computing Surveys 37*, 1 (2005), 42–81.

[23] Schlesinger, L., and Lehner, W. Scintra: A model for quantifying inconsistencies in grid-organized sensor database systems. In *Euro-Par* (2003), pp. 348–355.

[24] Spencer, H., and Lawrence, D. *Managing Usenet*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998.

[25] Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *ACM SIGCOMM* (2001), pp. 149–160.

[26] Thomas, R. H. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst. 4*, 2 (1979), 180–209.