

ON CONSISTENCY OF DATA IN STRUCTURED OVERLAY NETWORKS *

Tallat M. Shafaat[†], Monika Moser[‡], Ali Ghodsi[†], Thorsten Schütt[‡],
Seif Haridi[†], Alexander Reinefeld[‡]

Abstract Data consistency can be violated in Distributed Hash Tables (DHTs) due to inconsistent lookups. In this paper, we identify the events leading to inconsistent lookups and inconsistent responsibilities for a key. We find the inaccuracy of failure detectors as the main reason for inconsistencies. By simulations with inaccurate failure detectors, we study the probability of reaching a system configuration which may lead to inconsistent data. We analyze majority-based algorithms for operations on replicated data. To ensure that concurrent operations do not violate consistency, they have to use non-disjoint sets of replicas. We analytically derive the probability of concurrent operations including disjoint replica sets. By combining the simulation and analytical results, we show that the probability for a violation of data consistency is negligibly low for majority-based algorithms in DHTs.

1. Introduction

Peer-to-peer systems have gained tremendous popularity in recent years due to characteristics of scalability, fault-tolerance and self-management. Structured Overlay Networks (SONs) are a major class of these peer-to-peer system, examples of SONs include Chord [3], Chord# [6], Pastry [10] and DKS [7]. SONs provide lookup services for Internet-scale applications. Distributed Hash Tables (DHTs) use a SON's lookup service to provide a put/get interface for distributed systems with eventual consistency guarantees [11]. In contrast, many distributed systems require stronger consistency guarantees, relying on

*This research work is carried out under the SELFMAN project funded by the European Commission and the Network of Excellence CoreGRID funded by the European Commission.

[†]Royal Institute of Technology (KTH), {*tallat,haridi*}(at)kth.se

[‡]Zuse Institute Berlin (ZIB), {*moser,schuett,ar*}(at)zib.de

[†]Swedish Institute of Computer Science (SICS), *ali*(at)sics.se

services such as consensus [12] and atomic commits [13]. These services employ quorum techniques at their core to guarantee consistency.

Quorum based algorithms are not well-suited for DHTs. Quorum based techniques provide consistency guarantees as long as quorums overlap *i.e.* are never disjoint. On the contrary, the number of replicas of an item are not constant in a DHT. Hence, due to the extra replicas in a DHT, two quorums might not intersect, leading to inconsistent results.

Like most distributed systems, DHTs replicate a data item on different nodes in the SON to avoid losing data. In DHTs, the number of replicas may become greater than the replication degree for two reasons: *lookup inconsistencies* and *partitions*. Consider a DHT with replication degree three and an item replicated on nodes $N1, N2$ and $N3$. Due to lookup inconsistencies in the underlying SON¹, another node $N4$ might think that it is also responsible for the data item and will replicate the item. In such a case, a majority-based quorum technique [16] will result in inconsistent data as there are disjoint majority sets *e.g.* $\{N1, N2\}$ and $\{N3, N4\}$.

DHTs tolerate partitions in the underlying network by creating multiple independent DHTs. Due to consistent hashing [14], new nodes take responsibilities of inaccessible nodes and replicate data items. Thus, in the aforementioned case, if a partition occurs such that $N1, N2$ (partition P1) are separated from $N3$ (partition P2), owing to consistent hashing, replacement node $N3'$ will replicate the item in P1, and replacement nodes $N1'$ and $N2'$ will replicate the item in P2. This will result in the two partitions to have disjoint majority sets which will lead to data inconsistency.

It has been proved that it is impossible for a web service to provide the following three guarantees at the same time: consistency, availability and partition-tolerance [9]. These three properties have also been proved to be impossible to guarantee by a DHT working in an asynchronous network such as the Internet [7]. Thus, choosing to provide guarantees for two properties will violate the guarantee for the third. Since lookups are always allowed in DHTs, this implies DHTs are always available, thus consistency cannot be guaranteed.

In this paper, we study the causes and frequency of occurrence of lookup inconsistency under different scenarios in a DHT. We focus solely on lookup inconsistency leaving scenarios where complete partitions can happen, resulting in creation of multiple separate DHTs. We discuss and evaluate techniques that can be used to decrease the effect of lookup inconsistencies. Based on our simulation results while considering lookup inconsistencies to be the only reason for creation of extra replicas, we give an analytical model that gives the probability under which a majority-based quorum technique works correctly.

¹Informally, a lookup inconsistency means multiple nodes believe to be responsible for the same identifier. The term will be discussed in detail later.

Using techniques to decrease the effect of lookup inconsistency, we show that the probability of a quorum technique to produce consistent results is very high.

2. Background

Basics of a Ring-based SON. A SON makes use of an *identifier space*, which for our purposes is a range of integers from 0 to $N - 1$, where N is the length of the identifier space and is a large, fixed and globally known integer. For ring-based SONs, this identifier space is perceived as a ring by arranging the integers in ascending order and wrapping around at $N - 1$.

Every node in the system has a unique identifier drawn from the identifier space. Each node p has a pointer, *succ*, to its *successor*, which is the node immediately succeeding p , going in clockwise direction on the ring starting at p . Similarly, each node q has a pointer, *pred*, to its *predecessor*, which is the node immediately preceding q , going in anti-clockwise direction on the ring starting at q . To enhance routing performance, SONs also maintain additional routing pointers.

Handling Joins and Failures. Apart from *succ* and *pred* pointers, each node p also maintains a *successor-list* consisting of p 's c immediate successors, where c is typically set to $\log_2(n)$, n being the network size.

Chord [3] handles joins and failures using a protocol called *periodic stabilization*. Each node p periodically checks to see if its *succ* and *pred* are alive. If *succ* is found to be dead, it is replaced by the closest alive successor in the successor-list. If *pred* is found to be dead, p sets $\text{pred} := \text{nil}$.

Joins are also handled periodically. A joining node makes a lookup to find its successor s on the ring, and sets $\text{succ} := s$. Each node periodically asks for its successor's *pred* pointer, and updates its *succ* pointer if it gets a closer successor. Thereafter, the node notifies its current *succ* about its own existence, such that the successor can update its *pred* pointer if it finds that the notifying node is a closer predecessor than *pred*. Hence, any joining node is eventually properly incorporated into the ring.

Failure Detectors. SONs provide a platform for Internet-scale systems, aimed at working on an asynchronous network. Informally, a network is asynchronous if there is no bound on message delay. Thus, no timing assumptions can be made in such a system. Due to the absence of timing restrictions in an asynchronous model, it is difficult to determine if a node has actually crashed or is very slow to respond. This gives rise to wrong suspicions of failure of nodes.

Failure detectors are modules used by a node to determine if its neighbors are alive or dead. Since we are working in an asynchronous model, a failure

detector can only provide probabilistic results about the failure of a node. Thus, we have failure detectors working probabilistically.

Failure detectors are defined based on two properties: *Completeness* and *Accuracy* [5]. In a crash-stop model, completeness is the property that requires a failure detector to eventually detect as dead a node which has actually crashed. Accuracy relates to the mistake a failure detector can make to decide if a node has crashed or not. A perfect failure detector is accurate all the times, while the accuracy of an unreliable failure detector is defined by its probability of working correctly.

For our work, we use a failure detector similar to the baseline algorithm used by Zhuang et. al [4]. A node sends a ping to its neighbors at regular intervals. If it receives an acknowledgment within a timeout, the neighbor is considered alive. Not receiving an acknowledgment within the timeout implies the neighbor has crashed. The timeout is chosen to be much higher than the round-trip time between the two nodes.

3. Lookup and Responsibility Consistency

Data consistency is based on lookup consistency and responsibility consistency in the routing layer. We define these concepts and explain how a violation of these happens. The notion of a SON's configuration comprises the set of all nodes in the system and their pointers to neighboring nodes. A SON evolves by either changing a pointer, or adding/removing a node.

Lookup Consistency. *A lookup on a key is consistent, if lookups made for this key in a configuration from different nodes, return exactly the same node.*

Lookup consistency can be violated if some node's successor pointer does not reflect the current ring structure. Figure 1a illustrates a scenario, where lookups for key k can return inconsistent results. This configuration may occur if node $N1$ falsely suspected $N2$ as failed, while at the same time $N2$ falsely suspected $N3$ as failed. A lookup for key k ending at $N2$ will return $N4$ as the responsible node for k , whereas a lookup ending in $N1$ would return $N3$.

Responsibility. *A node n is said to be locally responsible for a certain key, if the key is in the range between its predecessor and itself, noted as $(n.pred, n]$. We call a node globally responsible for a key, if it is the only node in the system that is locally responsible for it.*

The responsibility of a node changes whenever its predecessor is changed. If a node has an incorrect predecessor pointer, it might have an overlapping range of keys with another node. However, to have concurrent operations on an item i with key k working on two different physical copies i' and i'' , the concurrent lookups should be inconsistent, and there should be an overlap of responsibility

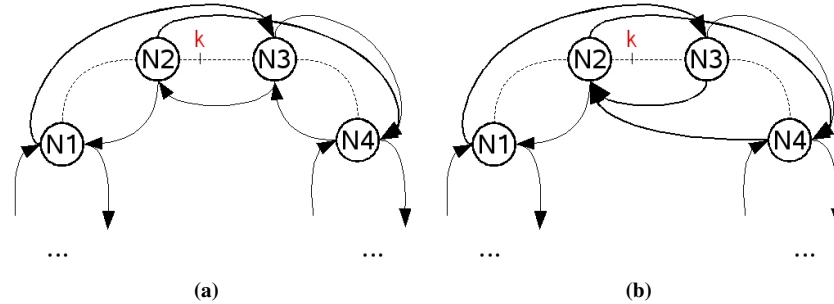


Figure 1: (a) *Lookup inconsistency* caused by wrong successor pointers. (b) *Responsibility inconsistency* caused by wrong successors and backlinks resulting in overlapping responsibilities.

for key k . Thus the following definition on responsibility consistency combines lookup consistency and global responsibility.

Responsibility Consistency. *The responsibility for a key is consistent if there is a globally responsible node for that key in the configuration.*

A situation where responsibility consistency for key k is violated is shown in Figure 1b. Here, lookup consistency for k cannot be guaranteed and both nodes, $N3$ and $N4$, are locally responsible for k . However, in Figure 1a, there is only one node $N3$ that is globally responsible despite lookup inconsistency. At node $N4$ the item is simply unavailable. The situation depicted in Figure 1b arises as the situation in Figure 1a with an additional wrong suspicion of node $N4$ about its predecessor $N3$.

As lookup consistency and responsibility consistency cannot be guaranteed in a SON it is impossible to ensure data consistency. However the violation of lookup consistency and responsibility consistency is a result of a combination of very infrequent events. In the following section we present simulation results that measure the probability of lookup inconsistencies and the probability of having an inconsistent responsibility, which turns out to be almost negligible.

4. Evaluation

In this section, we evaluate how often lookup inconsistencies and overlapping responsibilities occur. For our experiments, the measure of interest is the fraction of nodes that are correct, i.e. do not contribute to inconsistencies. The evaluations were done in a stochastic discrete event simulator in which we implemented Chord [3]. The simulator uses an exponential distribution for the inter-arrival time between events (joins and failures). To make the simulations scale, the simulator is not packet-level. The time to send a message from one node to another is an exponentially distributed random variable.

For our simulations, the level of unreliability of a failure detector is defined by its probability of working correctly. For the graphs, the probability of a *false positive*² is the probability of inaccuracy of failure detectors. Thus, a failure detector with a probability of false-positives equal to zero is a perfect failure detector.

In our experiments, we implemented failure detectors in two styles, *independent* and *mutually-dependent* failure detectors. For independent failure detectors, two separate nodes falsely suspect the same node as dead independently. Thus, if a node n is a neighbor of both m and o , the probability of m detecting n as dead is independent of the probability of o detecting n as dead. For mutually-dependent failure detectors, if a node n is suspected dead, all nodes doing detection on n will detect n as dead with higher probability. This may be similar to a realistic scenario where due to n or the network link to n being slow, nodes do not receive ping replies from n thus detecting it as dead. In the afore-mentioned case, if n is suspected, both m and o will detect it dead with higher probability than the probability of false-positive. Henceforth, we use independent failure detectors unless specified.

For our simulations, we first evaluate lookup inconsistencies for different degrees of false-positives. Next, we evaluate overlapping responsibilities in a system with and without churn. Furthermore, we compare lookup inconsistency and overlapping responsibilities. Finally, we present the results with mutually dependent failure detectors.

Our simulation scenario has the following structure: Initially, we successively joined nodes into the system until we had a network with 1024 nodes. We then started to gather statistics by regularly taking snapshots (earlier defined as a configuration) of the system. In each snapshot, we counted the number of correct nodes i.e. do not contribute to lookup inconsistency and overlapping responsibilities. For the experiments with churn, we introduced node joins and failures between the snapshots. We varied the accuracy of the failure detectors from 95% to 100%, where 100% means a perfect failure detector. This range seems reasonable, since failure detectors deployed on the Internet are usually accurate 98% of the time [4]. The results presented in the graphs are averages of 1800 snapshots and 30 different seeds.

Lookup Inconsistency. Figure 2a illustrates the increasing lookup inconsistency when the failure detector becomes inaccurate. The plot denoted ‘Total Inconsistencies’ shows the maximum over all possible lookup inconsistencies in a snapshot, whereas ‘Random Lookups’ shows the number of consistent lookups when – for each snapshot – lookups are made for 20 random keys, where each lookup is made from 10 randomly chosen nodes. If all lookups for

²detect an alive node as dead

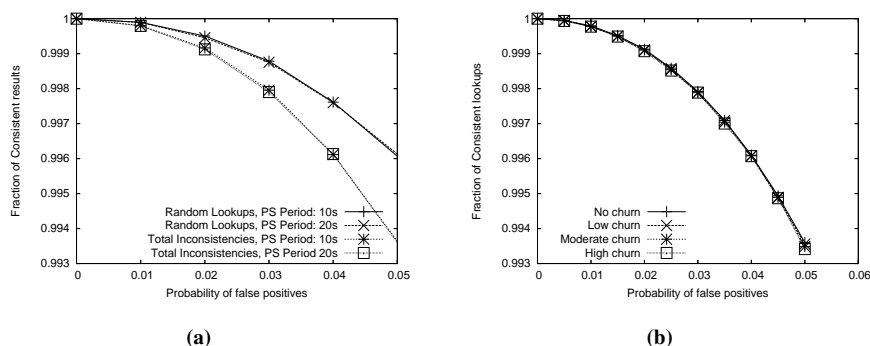


Figure 2: (a) Evaluation of lookup inconsistency. (b) Evaluation of lookup inconsistency under churn with only node joins.

the same key result in the same node, the lookup is counted as consistent. As can be seen, changing the periodic stabilization rate does not effect the lookup inconsistency in this case. This is due to the fact that there is no churn in the system.

Next, we evaluated lookup inconsistencies in the presence of churn. We varied the churn rate with respect to the periodic stabilization (PS) rate of Chord. For our experiments, we defined churn as node session times, to be in tens of minutes [15]. Short session times produce ‘high churn’ while long session times produce ‘low churn’. Figure 2b shows the results for our experiments. The Y-axis gives a count of the number of lookup inconsistencies per snapshot. As expected, churn does not effect lookup inconsistencies much. Though, even with a perfect failure detector (probability of false positive=0), there will be a non-zero though extremely low number of lookup inconsistencies given churn (2.79×10^{-7} for a high churn system). The reason is that an inconsistency in such a scenario only happens if multiple nodes join between two old nodes m, n (where $m.succ = n$) before m updates its successor pointer by running PS.

This effect of churn is due to node joins on lookup inconsistency can be reduced to zero if we allow lookups to be generated only from nodes that are fully in the system. A node is said to be *fully in the system* after it is accessible from any node that is already in the system. Once a node is fully in the system, it is considered to be in the system until it crashes. We define the first node which creates the ring as fully in the system.

Responsibility Inconsistency. Next, we evaluate the effect of unreliable failure detectors and churn on the responsibility consistency. The results of our simulations are presented in Figure 3a which shows that responsibility consistency is not effected by churn. Figure 3b shows that even with a lookup inconsistency, the chances of overlapping responsibilities are decreased

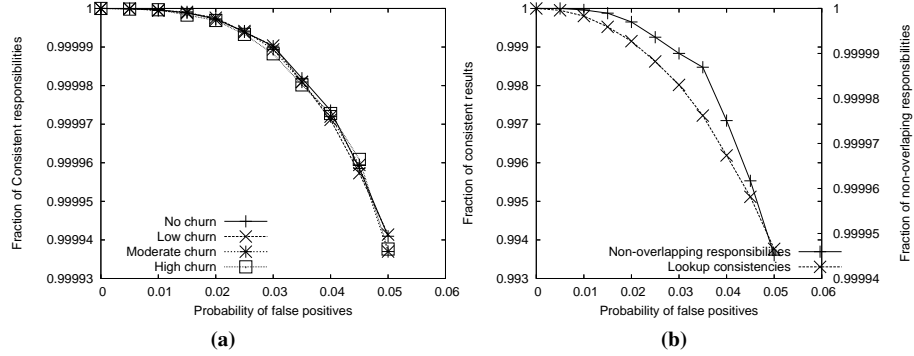


Figure 3: (a) Evaluation of overlapping responsibilities under churn with only node joins. (b) Comparison of lookup inconsistency and overlapping responsibilities. Lookup inconsistency is plotted against Y-axis while overlapping responsibilities is plotted against Y2-axis.

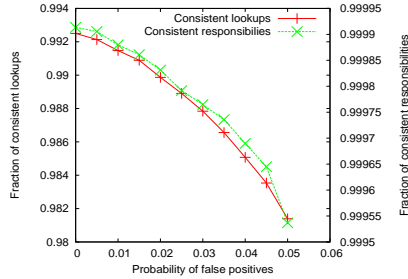


Figure 4: Evaluation of lookup inconsistency and overlapping responsibilities with mutually dependent failure detectors. Lookup inconsistency is plotted against Y-axis while overlapping responsibilities is plotted against Y2-axis.

roughly 100 times. This can be seen by the scale of the lookup inconsistency (Y-axis) and overlapping responsibility (Y2-axis).

Figure 4 shows results for a scenario without churn using mutually dependent failure detectors, where if a node n is suspected, the probability of nodes doing accurate detection on n drops to 0.7. In the scenario for the simulations, we suspect 32 random nodes. Compared to independent failure detectors, mutually dependent failure detectors produce higher lookup inconsistencies, but still low.

5. Data Consistency with Majority-Based Algorithms

To prevent loss of items stored in a SON, items are replicated on a set of nodes. The set of nodes that are responsible for the replicas, is determined by some replication scheme. Here we consider replication schemes that have a fixed replication factor r . An example for such a replication scheme is the DKS symmetric replication [2], where a globally known function determines

the set of keys under which replicas for an item are stored. The set of replicas for an item is called *replica set*.

In dynamic environments, some replicas might be temporary unavailable. However operations on an item should be able to succeed if they can access a subset of the replica set. Majority based algorithms require that at least a majority of replicas are available and tolerate the unavailability of the rest. Thus they are well suited for a SON. We refer to a set with a majority of replicas as a *majority set*. As each write operation includes such a majority set, two concurrent write operations have at least one replica in common, such that a conflict can be detected. It is crucial that the number of replicas in the system is never increased, otherwise one cannot guarantee that concurrent operations work on *non-disjoint majority sets*. However responsibility inconsistencies temporarily lead to an increase in the number of replicas. In the following, we analyze the probability of two concurrent operations working on *disjoint majority sets* given i inconsistencies in the replica set, to which we refer as *inconsistent replicas*.

Probability for Disjoint Majority Sets. In this section we model the probability that two operations work on disjoint majority sets in a given configuration. We assume that each responsibility inconsistency involves at most two nodes. More than two concurrent operations working on disjoint majority sets are not considered as the probability for it is considered as negligibly small.

The size of the smallest majority set is defined by $m = \lfloor \frac{r}{2} \rfloor + 1$. $T_{i,r}$, as shown in Equation (1), counts the number of all possible combinations for two majority sets, given $i > 0$ inconsistent replicas and the replication factor r . The formula takes into account the number of inconsistency j that are included in the majority sets. Each included inconsistency involves two possibilities to select a node that stores the replica.

$A_{i,r}$, in Equation (2), calculates the number of possible combinations for two disjoint majority sets, m_1 and m_2 , given $i > 0$ inconsistent replicas in the replica set and a replication factor r . We compute $A_{i,r}$ by choosing set m_1 and count all possible sets m_2 that are disjoint to m_1 . Part a of $A_{i,r}$ counts all possibilities to choose m_1 , such that at least one inconsistency is included. Again, j denotes the number of included inconsistencies. The second majority set m_2 shares k of the j inconsistencies (part b). For the remaining replicas of m_2 we have to consider how many inconsistencies l it will include from those that are left (part c).

$$T_{i,r} = \left(\sum_{j=\max(m-(r-i),0)}^{\min(i,m)} \binom{r-i}{m-j} \binom{i}{j} * 2^j \right)^2 \quad (1)$$

$$A_{i,r} = \sum_{j=lb_j}^{ub_j} \sum_{k=lb_k}^j \sum_{l=lb_l}^{ub_l} \overbrace{\binom{r-i}{m-j} \binom{i}{j} * 2^j}^a * \underbrace{\binom{j}{k} * \binom{(r-m)-(i-j)}{m-k-l} \binom{i-j}{l} * 2^l}_c \quad (2)$$

where (*ub* short for upper bound, *lb* short for lower bound)

$$lb_j = \max(1, m - (r - i))$$

$$ub_j = \min(i, m)$$

$$lb_k = \max(1, m - (r - m))$$

$$lb_l = \max(0, (m - k) - ((r - m) - (i - j)))$$

$$ub_l = \min(i - j, m - k)$$

$$pi_r = \sum_{i=1}^r (1-p)^{r-i} * p^i * \frac{A_{i,r}}{T_{i,r}} \quad (3)$$

pi_r calculates the overall probability in the system that two concurrent operations on one item operate on disjoint majority sets, where p is the probability of an inconsistency at a node as measured in our simulations.

Table 5 contains the probabilities for disjoint majority sets of two concurrent operations as calculated by $\frac{A_{i,r}}{T_{i,r}}$. As with an even replication factor the minimum number of replicas in common for two majority sets is two in contrast to one for an odd number of replicas, the probability for disjoint majorities is lower for an even number of replicas. In Figure 6 the results of the simulations are combined with Equation 3, where p denotes the simulated probability for an inconsistency. Depending on the failure detector accuracy it plots the probability to get non-disjoint majority sets. An even replication factor increases the probability of having non-disjoint majority sets, however less unavailable replicas can be tolerated.

6. Related Work

DHTs have been the subject of much research in recent years, with substantial amount of work on resilience of overlays to churn. While these studies show that overlays tolerate failures, they also show how lookups are effected by churn.

Rhea *et. al.* [1] have explored lookup inconsistencies for a real implementation under churn. Their approach differs from ours as they define a lookup to be consistent if a majority of nodes concurrently making a lookup for the same key get the same result. For our work, we require all results of making

r	$i=1$	$i=2$	$i=3$	$i=4$
1	0.5			
2	0	0.25		
3	0.16	0.31	0.42	
4	0	0.05	0.14	0.22
5	0.05	0.11	0.19	0.26

Figure 5: Probability for disjoint majority sets depending on the replication factor r and the number of inconsistencies i in the replica set.

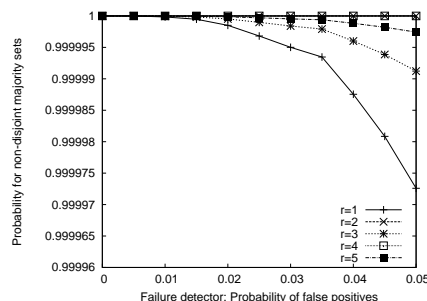


Figure 6: Probability for non-disjoint majority sets in a SON, depending on the accuracy of the failure detector

the lookup for the key to be the same. Furthermore, our work is extended to responsibility consistency. In their work, Rhea *et. al.* also study lookup inconsistency in an implementation of Pastry [10] called FreePastry [18], while we experiment with Chord.

Liben-Nowell *et. al.* [17] study the evolution of Chord under churn. Their study is based on a fail-stop model *i.e.* they assume perfect failure detection and reliable message delivery. Consequently, they ignore “false suspicions of failure”, which is the main topic of our study as we observe that imperfect failure detectors are the main source of lookup inconsistencies.

Zhuang *et. al.* [4] studied various failure detection algorithms in Overlay Networks. They also use the same approach as Rhea *et. al.* [1] to define inconsistencies, which differs from our work.

7. Conclusion

This paper presents an evaluation of consistency in SONs. Data consistency cannot be achieved if responsibility consistency is violated. We describe why it is impossible to guarantee responsibility consistency in SONs. By simulating a Chord SON, we show that the probability of violating responsibility consistency is negligibly low.

We analytically derive the probability that majority-based operations are working on non-disjoint majority sets given an inconsistent responsibility for at least one replica. Operations that work on disjoint majority sets lead to inconsistent data. By combining the results from simulations and analysis, we show that the probability for getting inconsistent data when using majority based algorithms is significantly low. Furthermore, we conclude that since the accuracy of the failure detector greatly influences lookup and responsibility consistency, significant attention should be paid to the failure detection algorithm.

References

- [1] S. Rhea, D. Geels, T. Roscoe and J. Kubiatowicz. Handling Churn in a DHT. In *Proceedings of USENIX Annual Technical Conference*, 2004 Berkeley
- [2] A. Ghodsi, L. Onana Alima and S. Haridi. Symmetric Replication for Structured Peer-to-Peer Systems. *DBISP2P*, 2005, Trondheim, Norway
- [3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. newblock *IEEE/ACM Transactions on Networking (TON)*, 11(1):17.32, 2003.
- [4] S.Q. Zhuang, D. Geels, I. Stoica, R.H. Katz. On Failure Detection Algorithms in Overlay Networks. In *Proceedings of INFOCOM'05*, Miami, 2005
- [5] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43:2, 1996
- [6] T. Schütt, F. Schintke and A. Reinefeld. Structured Overlay without Consistent Hashing: Empirical Results. In *Proceedings of GP2PC'06*, 2006
- [7] A. Ghodsi. Distributed k -ary System: Algorithms for Distributed Hash Tables. *PhD Dissertation*, KTH—Royal Institute of Technology Oct, 2006
- [8] M. Moser, S. Haridi. Atomic Commitment in a Transactional DHT. In *Proceedings of the CoreGRID Symposium*, 2007
- [9] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In *SIGACT News*, 2002
- [10] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of MIDDLEWARE.01*, volume 2218 of Lecture Notes in Computer Science (LNCS), Germany, 2001
- [11] F. Dabek. A Distributed Hash Table. *Doctoral Dissertation*, MIT — Massachusetts Institute of Technology, 2005
- [12] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, ACM Press, 1998, 16, 133-169
- [13] N. Lynch, M. Merritt, W. Weihl, and A. Fekete. Atomic Transactions. Morgan Kaufmann Publishers, 1994
- [14] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, El Paso, 1997
- [15] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proceedings of SOSP*, 2003.
- [16] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of SOSP '79*, New York, USA, 1979
- [17] D. Liben-Nowell, H. Balakrishnan, D. Karger. Analysis of the Evolution of Peer-to-Peer Systems In *Proceedings of PODC '02*, USA, 2002
- [18] Freepastry. <http://freepastry.rice.edu/>