Key-Based Consistency and Availability in Structured Overlay Networks

Tallat M. Shafaat
Royal Institute of Technology (KTH)
Stockholm, Sweden
tallat@kth.se

Monika Moser, Thorsten Schütt and Alexander Reinefeld
Zuse Institute Berlin
Berlin, Germany
{moser,schuett,ar}@zib.de

Ali Ghodsi and Seif Haridi
Swedish Institute of Computer Science
Stockholm, Sweden
{ali,seif}@sics.se

ABSTRACT

Structured Overlay Networks provide a promising platform for high performance applications since they are scalable, fault-tolerant and self-managing. Structured overlays provide lookup services that map keys to nodes that can be used as processing or storage resources. The lookups for a key may return inconsistent results. Consequently, it is nontrivial to provide consistent data services on the top of structured overlays that are built on key-based search. In this paper, we study the frequency of occurrence of inconsistent lookups. We show that the effect of lookup inconsistencies can be reduced by assigning responsibility of key intervals to nodes. We present our results as a trade-off between consistency and availability of keys. Further, since many distributed applications employ quorum techniques at their core, we analyze the probability that majority-based quorum techniques will function correctly in a structured overlay with inconsistent lookups. Our analysis shows that the probability of majority-based algorithms to function correctly despite lookup inconsistencies is high.

1. INTRODUCTION

Structured Overlay Networks, such as Chord [13] and DKS [3], form a major class of peer-to-peer systems. Structured overlays provide lookup services for Internet-scale applications, where a lookup maps a key to a node in the system. The node mapped by the lookup can then be used for data storage or processing. Distributed Hash Tables (DHTs) [3] use an overlay's lookup service to store data and provide a put/get interface for distributed systems. Since structured overlays are "best-effort", DHTs built on these overlays typically guarantee eventual consistency. In contrast, many distributed systems, such as distributed file systems and distributed databases , require stronger consistency guarantees. These systems generally rely on services such as consensus and atomic commit.

DHTs are designed to cope with high rates of churn (node joins and leaves). Due to consistent hashing [6] in a DHT, existing nodes take over key responsibilities of inaccessible nodes, and newly joined nodes take over a fraction of the responsibilities of existing nodes. Similarly, DHTs tolerate partitions in the underlying network by creating multiple independent DHTs and provide availability for all keys in each DHT.

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 [5]. These three properties have also been proved to be impossible to guarantee by a DHT working in an asynchronous network such as the Internet [3]. Thus, choosing to provide guarantees for two properties will violate the guarantee for the third. In this work, we focus on availability and consistency while assuming there is no network partition.

As we discuss in section 3, inconsistent data in DHTs mainly arises due to inconsistent lookups. In this paper, we study the causes and frequency of occurrences of lookup inconsistencies under different scenarios in a DHT. We discuss and evaluate techniques that can be used to decrease the effect of lookup inconsistencies. We show how decreasing the effect of lookup inconsistencies affects availability. Based on our simulation results, we give an analytical model that gives the probability under which a majority-based quorum technique works correctly. Using techniques to decrease the effect of lookup inconsistency, we show that we can achieve key consistency with high probability.

Outline: First, we define the DHT model that our work is based on in Section 2. Section 3 introduces lookup consistency and explains how it can be violated. Section 4 explains techniques that can be used to reduce consistency violation. Simulations which study the probability of a violation of lookup consistency and the affect of techniques to reduce inconsistency are presented in Section 5. In Section 6 we discuss related work. Finally, Section 7 presents the conclusion of our work.

2. BACKGROUND

Ring-based DHT: A DHT makes use of an *identifier space*, which for our purposes is defined as a set of integers $\{0, 1, \cdots, \mathcal{N}-1\}$, where $\mathcal{N}$ is some apriori fixed, large, and globally known integer. This identifier space is perceived as a ring that wraps around at $\mathcal{N}-1$. Every node in the system, has an unique identifier from the identifier space. Each node keeps a pointer *succ* to its *successor* (first node met going clockwise) and a pointer *pred* to its *predecessor* (first node

INFOSCALE 2008, June 4-6, Vico Equense, Italy
 978-963-9799-28-8
DOI 10.4108/ICST.INFOSCALE2008.3537

met going anti-clockwise) on the ring. Ring-based DHTs also maintain additional routing pointers on top of the ring to enhance routing.

We choose Chord [13] for our analysis, which is one of the most popular ring-based overlay. Chord handles joins and failures using a protocol called *periodic stabilization*. The protocol works such that each node n should eventually point to the first node clockwise from n as *succ* and the first node anti-clockwise from n as *pred*.

Failure Detectors: DHTs 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. Since timing assumptions can not be made in asynchronous networks, it is difficult to determine if a node has crashed or is very slow to respond. This gives rise to inaccurate suspicion of node failure. Thus, failure detectors - modules used by a node to determine if another node is alive of dead - work probabilistically.

Failure detectors are defined based on two properties: *completeness* and *accuracy* [2]. In a crash-stop process model, completeness requires the failure detector to eventually detect all crashed nodes. Accuracy relates to the mistake a failure detector can make to decide if a node has crashed or not.

3. CONSISTENCY VIOLATION

In this section, we show how lookup inconsistencies may arise and discuss how lookup inconsistencies can lead to data inconsistency. Unless specified, the term consistency refers to lookup consistency, otherwise, we explicitly say data consistency. A configuration of the DHT is a set of all nodes and their pointers to neighboring nodes. A DHT evolves by either changing a pointer, or adding/removing a node.

Definition 1. *A configuration of the system is consistent if, in that configuration, lookups made for the same key from different nodes, return the same node.*

In a configuration where consistency is violated, we have inconsistent lookups *i.e.* multiple lookups for the same key may return different nodes in that configuration. Lookup consistency can be violated if some node's successor pointers do not reflect the current ring structure. Figure 1(a) illustrates such a configuration where lookups for key k can return inconsistent results. This configuration arises when, due to inaccuracy of the failure detector, $N1$ falsely suspects $N2$ and $N3$ as failed. Thus, $N1$ believes that the next (clockwise) alive node on the ring in $N4$, so it points its successor pointer to $N4$. Subsequently, a lookup for key k ending at $N1$ will return $N4$ as the responsible node for k, whereas a lookup ending in $N2$ would return $N3$.

In the scenario depicted in figure 1(a), an update for the data stored under key k will be stored at either $N3$ or $N4$. A read for data at k will return inconsistent/old results if it reaches the node that didn't receive the update.

4. REDUCING INCONSISTENCIES

In this section, we discuss two techniques to reduce lookup inconsistencies: (1) Local responsibilities (2) Quorum-based algorithms. These techniques can be used separately, or together to get the best results.

4.1 Local Responsibilities

Definition 2. *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 configuration that is* locally responsible *for the key.*

The responsibility of a node changes whenever its predecessor is changed. As can be noted, a configuration is consistent if there is a globally responsible node for each key. Similarly, the responsibility for a key k is consistent if there is a node globally responsible for k.

We modify the lookup operation of a such that a lookup always returns from the locally responsible node. Thus, before returning the result of a lookup, the node checks if it is locally responsible for the key being looked up. In case the node is not locally responsible, it can either forward the request to its predecessor or ask the initiator of the lookup to retry.

Although the configuration depicted in figure 1(a) is inconsistent, yet it is consistent with respect to local responsibilities. This is because, instead of replying, the lookup for k at $N1$ will be forwarded to $N4$. Since $N4$ is not locally responsible for k, it will not reply. On the other hand, the lookup at $N2$ will be forwarded to $N3$ which will reply as it is locally responsible for k. Thus, updates and reads for data items stored under key k will give consistent results.

If a node has an incorrect predecessor pointer, the range of keys it is responsible for can overlap with another node's key range. In such a case, there will be multiple nodes responsible for the same key leading to inconsistency. This can be seen in figure 1(b). Here, both $N3$ and $N4$ are locally responsible for k. This situation may arise if $N1$ falsely suspects $N2$ while both $N2$ and $N4$ falsely suspect $N3$.

Figure 1(b) shows that the method of local responsibilities does not completely solve the problem of inconsistencies, but it decreases inconsistencies. This is mainly because without local responsibility, only one node doing inaccurate failure detections is enough to introduce inconsistencies, while multiple nodes have to do simultaneous inaccurate failure detections to introduce responsibility inconsistencies

4.1.1 Key Availability

Unfortunately, as a side effect, local responsibilities give rise to keys being unavailable.

Definition 3. *In a configuration, a key k is available if there exists a reachable node n such that n is locally responsible for k.*

Here, a node n is *reachable* in a configuration if there exists a node m such that n is the successor of m, *i.e.* $m.succ = n$ and $n \neq m$.

Availability of a key is affected by both churn and inaccurate failure detectors. When a node joins the system, it changes the responsibilities of its successor. This leads to temporary unavailability of some keys. Figure 1(c)(i) shows a configuration when $N2$ joins the overlay. $N3$ points to $N2$ as its predecessor thus making $k1$ unavailable. Key $k1$ remains unavailable until $N1$ runs periodic stabilization and sets $N1.succ = N2$ and $N2$ sets $N2.pred = N1$.

Similarly, failure of a node leads to temporary unavailability of keys until the failure is detected. Such a case is shown in figure 1(c)(ii) where $N2$ crashes. Key $k2$ remains unavailable until $N1$ detects failure of $N2$ and sets $N1.succ = N3$ and $N3$ sets $N3.pred = N1$.

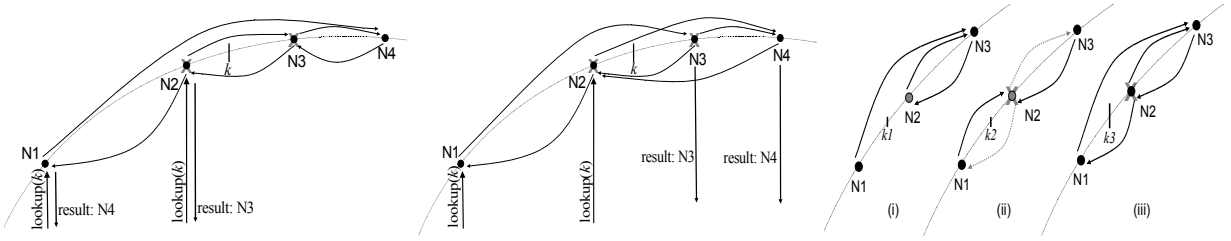


Figure 1: **(a) An inconsistent configuration. Due to imperfect failure detection, $N1$ suspects $N2$ and $N3$, thus pointing to $N4$ as successor. (b) An inconsistent configuration with respect to *local responsibilities*. $N1$ falsely suspects $N2$ while $N2$ and $N4$ falsely suspect $N3$. (c) Unavailability of key when a node (i) $N2$ joins (ii) $N2$ fails (iii) $N1$ falsely suspects $N2$ and updates its successor.**

Inaccuracy of failure detectors also leads to unavailability of keys. This occurs when a node falsely suspects its successor and removes its pointer to the suspected node. Keys for which the suspected node is responsible will temporarily become unavailable. Such a scenario is shown in figure 1(c)(iii) where $N1$ suspects $N2$ leading to unavailability of $k3$ as $N2$ becomes unreachable.

Systems that implement atomic join and graceful leaves such as DKS [3] will alleviate the case in fig. 1(c)(i), but not cases shown in fig. 1(c)(ii) and fig. 1(c)(iii).

4.2 Quorum-based Algorithms

Like most distributed systems, DHTs replicate data on different nodes to increase availability and prevent loss of data. Some examples of replication in DHTs include successor list replication [13] and key-based replication such as symmetric replication [3]. In what follows, we assume key-based replication, where an item is stored under various keys [3].

The basic idea of quorum-based algorithms is that conflicting operations acquire a sufficient number of votes from different replicas such that they have at least one intersection at one replica. Gifford introduced an algorithm for the maintenance of replicated data that uses weighted votes [4]. In our work, we consider majority-based algorithms which are a special case of quorum algorithms. The reason for choosing majority-based quorum algorithms (MBQAs) is that they are most robust and widely used form of quorum algorithms, e.g. in group membership, concurrency control and non-blocking atomic commit. In a MBQA, every replica is assigned exactly one vote and every operation has to collect at least a majority of votes (called a *majority set*). Quorum techniques can be used separately on the data-level as well to reduce data inconsistencies, yet our focus is to show how to use these techniques to reduce the affect of routing inconsistencies which will in-effect reduce data inconsistencies in DHTs. As we discuss shortly, using replicas and majorities distributes the problem of lookup inconsistency over all replicas.

4.2.1 Key-based Consistency with MBQAs

Consider a DHT with replication degree three. A data item to be stored under key k is thus stored under keys $\{k1, k2, k3\}$. Say nodes $N1$ and $N3$ are responsible for $k1$ and $k3$, while due to lookup inconsistency, two nodes $N2, N2'$ are responsible for $k2$ [1]. Any update or read for k has to operate on a majority *i.e.* two nodes in this case. Consistency in the afore-mentioned case depends on the way we choose majorities. Figure 2(a) shows a case where majorities for multiple updates overlap, thus only one update

[1] just like $N3$ and $N4$ are responsible for k in figure 1(b)

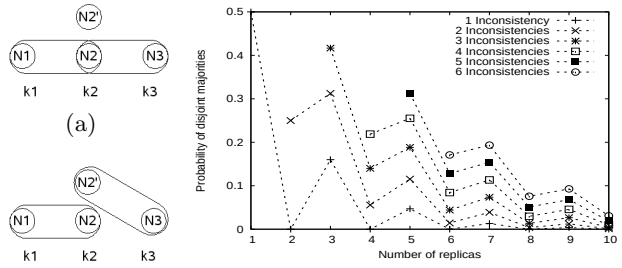


Figure 2: **(a) Example with intersecting majority sets. (b) Example with non-intersecting majority sets. (c) Probability of getting disjoint majority sets (Y-axis) for a replica set given replica degree (X-axis) and the number of lookup inconsistencies for the keys in the replica set.**

succeeds and the data remains consistent. On the other hand, figure 2(b) shows a case where the majorities do not overlap, hence updates will happen on different majority sets thus creating data inconsistent. Using MBQAs in DHTs increases the chances of consistency since even with lookup inconsistencies, multiple majorities exist that will lead to data consistency.

Probability model for disjoint majority sets: In this section, we use the counting principle to analytically derive the probability that two operations work on disjoint/non-overlapping majority sets given the system configuration is the same for the two operations. The probability of disjoint majority sets is the ratio between the number of possible disjoint majority sets and the number of all combinations of majority sets that two operations in one configuration can include. We assume that for a responsibility inconsistency in the configuration, only two nodes are responsible for the inconsistency.

$$A_{i,r} = \sum_{j=\max(m-r+i,0)}^{\min(m,i)} \sum_{k=\max(2m+i-r-2j,0)}^{\min(m,i-j)} 2^{k+j} \binom{r-i}{m-j}\binom{i}{j} \binom{r-m-i+2j}{m-k}\binom{i-j}{k} \quad (1)$$

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

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

Consider a DHT with replication degree r (where $r > 0$), a configuration with i number of responsibility inconsistencies (where $i > 0$) and size of the smallest majority set m (where $m = \lfloor \frac{r}{2} \rfloor + 1$). $T_{i,r}$ (eq. (2)) gives the number of all possible combinations for two majority sets. Here, j is the number

of inconsistencies j included in a majority set. Since each inconsistency creates two possibilities to select a node, we multiple with 2^j.

$A_{i,r}$ (eq. (1)) gives the number of possible combinations for two disjoint majority sets $mset1$ and $mset2$. We compute $A_{i,r}$ by choosing $mset1$ and calculating every possible $mset2$ that is disjoint to $mset1$. j denotes the number of inconsistencies that are included by $mset1$. $mset2$ can share a subset of these j inconsistencies and additionally include up to $i - j$ remaining inconsistencies. The derived formula is similar to a hyper-geometric distribution.

Assuming inconsistencies are independent, pi_r calculates the probability that two subsequent operations in one configuration work on disjoint majority sets, where p is the probability of an inconsistent responsibility.

Figure 2(c) plots the probability of having disjoint majority sets pr for two operations as it is calculated by $\frac{A_{i,r}}{T_{i,r}}$. It shows how pr depends on the system's replication factor r and on the number of inconsistencies i in the replica set. An important observation is that an even replication degree reduces pr considerably. The reason for such a behaviour is that for majority-based quorums with even replication degree, any two quorums overlap over at least two replicas (say r1 and r2). Due to lookup inconsistency, even if quorums don't overlap at r1, there is a significant chance that they will overlap at r2. This reduces the probability of getting disjoint majority sets.

As lookup consistency cannot be guaranteed in a DHT, even with using local responsibilities and quorum techniques, it is impossible to ensure data consistency. However the violation of lookup consistency when using the afore-mentioned techniques is a result of a combination of very infrequent events which is evaluated in the following section.

5. EVALUATION

In this section, we evaluate the frequency of occurrence of lookup inconsistencies, overlapping responsibilities and unavailability of keys resulting from unreliable failure detectors and churn. The measure of interest is the fraction of nodes that are correct, i.e. do not contribute to inconsistencies and the percentage of keys available. The evaluations are done for a network size of 1000 nodes in a stochastic discrete event simulator in which we implemented Chord [13].

For our simulations, we employ failure detectors that are complete but not accurate. The level of reliability of a failure detector is defined by its probability of working correctly. For the graphs, the probability of a *false positive* (detect an alive node as dead) is the probability of inaccuracy of failure detectors. We implemented failure detectors in two styles: *independent* and *mutually-dependent*. For independent failure detectors, two separate nodes falsely suspect the same node as dead independently. For mutually-dependent failure detectors, if a node p is suspected dead, all nodes doing detection on p will detect p as dead with higher probability, representing a positive correlation between suspicions of different failure detectors. This may be similar to a realistic scenario where due to p or the network link to p being slow, nodes do not receive ping replies from p thus detecting it as dead. Unless specified, we use independent failure detectors. For our experiments, we varied the accuracy of the failure detectors from 95% to 100% which is a reasonable range [14].

Lookup inconsistencies: Figure 3(a) illustrates the increase in lookup inconsistencies with inaccuracy of failure detectors and churn. As the figure shows, churn does not effect lookup inconsistencies much. Even with a perfect failure detector (false positive=0), there will be a non-zero though extremely low number of lookup inconsistencies given churn. An inconsistency in such a scenario happens if multiple nodes join between two old nodes m, n (where $m.succ = n$) before m updates its successor pointer by running periodic stabilization.

Affect of local responsibilities: Next, we evaluate the effect of unreliable failure detectors and churn on responsibility consistency. The results of our simulations (omitted due to space constraints) show that responsibility consistency is also not effected by churn. Our results for unreliable failure detectors are shown in Figure 3(b) (Y2-axis).

Figure 3(b) also shows that given a lookup inconsistency, the probability of overlapping responsibilities is only 0.01. This can be seen by the scale of the lookup inconsistency (Y-axis) and overlapping responsibility (Y2-axis).

Mutually-dependent failure detectors: Figure 3(c) shows results for a scenario without churn using mutually-dependent failure detectors, where if a node n is suspected, the accuracy of all failure detectors detecting n falls to 70%. In the scenario for the simulations, we suspect 32 random nodes. Compared to independent failure detectors, mutually dependent failure detectors produce higher lookup inconsistencies.

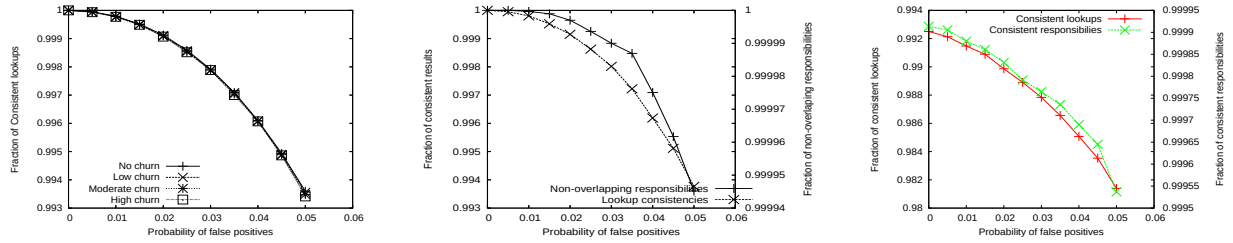
Key availability: Next, we evaluated the percentage of keys available in a system with churn and inaccurate failure detectors. Experimental studies [12] show that lifetime of nodes staying in the system ranges from tens of minutes to more than an hour. Further, experiments show that where node's mean lifetime is 1 hour, the optimal freshness threshold for periodic stabilization is about 72s [7]. Consequently, for our experiments, we choose a stabilization rate of 1 minute and varied the lifetime of nodes in tens of minutes. The results for our experiments are shown in figure 4(a), which shows that availability is effected by both inaccuracy of failure detectors and churn. Also, even with perfect failure detectors, churn results in unavailability of keys.

The affect of churn on key availability can be reduced by using atomic ring maintenance algorithms [3] [11]. These algorithms give a consistent view of the ring in the presence of node joins and leaves by transferring responsibilities of keys before a join or leave completes.

Affect of MBQA: By substituting p in Equation 3 with the simulation results using local responsibilities, we get the results illustrated in Figure 4(b). Figure 4(b) shows the probability for two non-disjoint majority sets in a certain configuration of a DHT depending on the probability that a failure detector makes false positives. Reflecting the results of figure 2(c), the probability that two majority sets are disjoint in a system with an even number of replicas is almost zero. However, in such as system, lesser number of unavailable replicas can be tolerated for an operation to succeed.

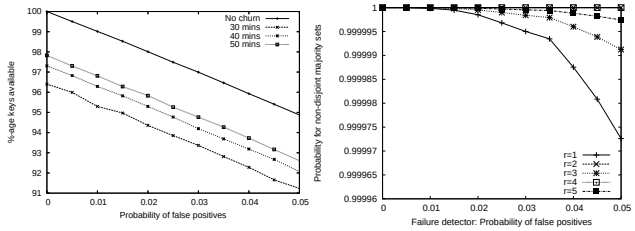
6. RELATED WORK

An important design goal for distributed systems is to provide data consistency. Since DHTs are aimed to work over asynchornous networks with high rate of churn, providing consistency in DHTs becomes an intersting and nontrivial problem. The problem at hand can be attacked on two levels: *routing level* and *data level*. We focus on the routing



(a) (b) (c)

Figure 3: **(a) Evaluation of lookup inconsistency under churn, with only node joins. (b) Comparison of lookup inconsistency (Y-axis) and overlapping responsibilities (Y2-axis). (c) Comparison of lookup inconsistency (Y-axis) and overlapping responsibilities (Y2-axis) with mutually dependent failure detectors.**



(a) (b)

Figure 4: **(a) Evaluation of keys availability under churn generated by different lifetime of nodes. (b) Probability of disjoint majority sets for two majority based operations in the same configuration.**

level by providing techniques to reduce the affect of lookup inconsistencies. Solutions on the data level (e.g. [9]) might have constraints or depend on the application and semantics of data operations.

Atomic ring maintenance algorithms [3, 8, 11] provide lookup consistency under joins and leaves, ignoring failures and inaccurate failure detectors. As we have shown, the main contributors to lookup inconsistency are inaccurate failure detectors, which is the focus of our work.

There has been work done on studying lookup inconsistencies under churn. Rhea *et. al.* [10] have explored lookup inconsistencies for Chord. Their work overlooks the fact that imperfect failure detectors mainly cause inconsistent lookups.

Bhagwan *et. al.* [1] attack the problem of availability in peer-to-peer systems. Contrary to our work, they focus on availability of hosts and thus data stored at the hosts. Since we are working on the routing level, we focus on availability of keys and thus nodes responsible for keys.

7. CONCLUSIONS [2]

We studied the frequency of lookup inconsistencies and found that its main cause is inaccurate failure detectors. Hence, choice of a failure detection algorithm is of crucial importance in DHTs.

While effects of lookup inconsistencies can be reduced by using local responsibilities, we show that using responsibility of keys may affect availability of keys. This is a trade-off between availability and consistency. Many data dependent applications may prefer unavailability to inconsistency.

We show that using quorum-based techniques amongst replicas of data items further reduce lookup inconsistencies. Since majority-based quorum techniques require a majority of the replicas to make progress, these algorithms may still make progress even with unavailability of some keys/nodes. Thus, using a combination of local responsibilities and quorum techniques is attractive in scalable applications.

Due to the dynamics and decentralization of DHTs, it is difficult to build abstractions with stronger consistency guarantees on top of DHTs. We propose using techniques on the routing level to decrease data inconsistencies. These techniques can be used with techniques at the data level to get best results. Our results show that it is reasonable to build reliable services on top of a DHT. In our future work, we plan to evaluate an implementation of a transactional storage service on top of a DHT using routing-level techniques described in this paper.

[2] **THIS WORK HAS BEEN FUNDED BY THE EUROPEAN UNION IN THE SELFMAN PROJECT (CONTRACT 034084) AND COREGRID NETWORK OF EXCELLENCE (CONTRACT 004265).**

8. REFERENCES

[1] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[2] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43, 1996.

[3] A. Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD thesis, KTH — Royal Institute of Technology, Sweden, Dec. 2006.

[4] D. K. Gifford. Weighted voting for replicated data. In *SOSP '79: Proceedings of the seventh ACM symposium on Operating systems principles*, pages 150–162, New York, NY, USA, 1979. ACM Press.

[5] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[6] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and 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*, 1997.

[7] J. Li. *Routing tradeoffs in dynamic peer-to-peer networks*. PhD thesis, MIT — Massachusetts Institute of Technology, Nov. 2005.

[8] P. Linga, A. Crainiceanu, J. Gehrke, and J. Shanmugasudaram. Guaranteeing correctness and availability in p2p range indices. In *Proceedings of 2005 ACM SIGMOD*, pages 323–334, 2005.

[9] N. A. Lynch, D. Malkhi, and D. Ratajczak. Atomic data access in distributed hash tables. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 295–305, London, UK, 2002. Springer-Verlag.

[10] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. Technical report, EECS Department, University of California, 2003.

[11] J. Risson, K. Robinson, and T. Moors. Fault tolerant active rings for structured peer-to-peer overlays. *lcn*, 0:18–25, 2005.

[12] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *In Proc. of MMCN*, 2002.

[13] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. of the ACM SIGCOMM*, 2001.

[14] S. Zhuang, D. Geels, I. Stoica, and R. Katz. On failure detection algorithms in overlay networks. In *Proc. of INFOCOM*, 2005.