CS87 Project Report: Adaptive Routing For DHTs

Do June Min, Zheyuan Ryan Shi, Yu Jian Wu
Computer Science Department, Swarthmore College, Swarthmore, PA 19081
{dmin1, zshi1, ywu1}@swarthmore.edu

July 19, 2019

Abstract

The Distributed Hash Table (DHT) has seen great theoretical progress and wide applications in practice, such as peer-to-peer file sharing, content delivery network, and distributed caching. In practice, nodes in a DHT often have different computational capacity. This heterogeneity is dynamic due to the nodes being non-dedicated resources and the absence of a centralized scheduler. To address these issues, we study the routing algorithm in Kademlia, the most widely used DHT. We make the following contributions. (i) Complement to the XOR metric in Kademlia, we propose a new metric on the network topology which incorporates real-time load and latency. Hence the heterogeneity of the nodes are accounted for. (ii) We devise an efficient mechanism to measure the computing load on other nodes in the DHT, which addresses the dynamic nature of the capacity. (iii) We perform extensive experiments to demonstrate the merit of our enhanced routing algorithm in practice.

1 Introduction

A Distributed Hash Table (DHT) provides (key, value) storage and look-up functions in a distributed manner. Research efforts in DHT have led to successful applications in various domains, ranging from file sharing [12, 2] and Domain Name Service [13], to Content Delivery Network [4] and instant messaging communication [11]. In a DHT, all participating users (nodes) collectively maintains a hash table of data in the form of (key, value) pairs. A key component of a DHT system is the routing algorithm during the loop-up, which directly impacts the DHT's efficienty. Such algorithms in existing DHT models typically traverse through $O(\log n)$ nodes, where n denotes the total number of nodes in the system. While such complexity is decent in theory, these routing algorithms ignore the heterogeneous and dynamic nature of the system, which renders the supposedly optimal route suboptimal in practice. For example, if a DHT has a few Intel Pentium 4 nodes with the rest being Intel Core i7 nodes, it may not be ideal to route much traffic through the Pentium 4 ones. Such capacity difference can be dynamic. Because of the decentralized nature, DHT has little control over the other processes running on its nodes. If some nodes in the DHT are training some neural networks, avoiding them *temporarily* is reasonable. Such issues are common in practice, and they call for a routing protocol which is adaptive to real-time node capacity. In the next section, we detail how existing work does not adequately address this problem. In what follows we outline our approach to fill the gap.

Adaptive routing protocol Our solution is based on Kademlia [10], the most widely used DHT model in practice. In Kademlia, the distance between two nodes is measured by the XOR value of their node IDs. As a result, the routing protocol is a greedy algorithm based on this distance. To incorporate real-time capacity information, we propose a new distance which

is a weighted average of the XOR distance, the destination's idle CPU power, and the destination's idle memory capacity. Our framework is general enough such that additional information or different functional form of the distance can be easily accomodated.

Real-time load information The adaptive routing protocol as described above requires each node know the load information of some other nodes. We devise an efficient mechanism where each node regularly logs its capacity and send it to other nodes upon request. This mechanism is integrated into the general DHT framework and thus brings little overhead.

Simulation We evaluate our ideas using two numerical simulations. First, we measure the performance of DHT look-up using our adaptive routing protocol in Kademlia. We also carried out conceptual evaluation on a large, virtual, DHT. In both experiments, we analyze the results and provide the rationale of different outcomes.

2 Related Work

Distributed hash tables provide a way to access data distributed over multiple nodes inside a network [19]. Many DHT models have been proposed, with some notable ones being Chord [18], Pastry [16], Tapestry [21], and Kademlia [10]. All of them provides a loop-up routing algorithm which traverses only $O(\log n)$ nodes assuming there are n nodes in the DHT. However, such guarantee alone is inadequate for DHT in practice. For example, an important task that must be addressed in a practical DHT system is minimizing the impact of newly joining/quitting nodes. To achieve this, *structure-based hashing* methods impose a topological structure on the nodes, such as rings [18], cubes [14], or trees [16, 21, 10]. These structures introduce the concept of *distance* between different nodes, which then can be used to partition the keyspace, allowing for efficient look-ups. This look-up would amount to a linear search of the node-space. However, *finger tables* reduce the complexity of look-ups to $O(\log n)$ by providing a way for search queries to hop between nodes that are far away from each other.

The focus of our paper is on the improvement of the routing algorithm. Ideally, the routing in DHT should avoid creating routing hot spots, i.e. uneven traffic across the nodes [15]. Furthermore, in the actual deployment of DHT, nodes have different capacity in dealing with routing and queries [17]. A routing protocol which ignores this can exacerbate the hot spot problem and compromise the overall performance.

Since the advent of the aforementioned DHT models, a wealth of literature has focused on this issue. Zhang et al. [20] incorporates the underlying network latency into the routing process. Other works consider the latency inside each node, which is also the problem we study. Cuevas et al. [3] propose a simple modification to the Chord's routing protocol, which significantly alleviates the hot spot problem, yet their analysis is mostly about homogeneous nodes. Godfrey and Stoica [5] and Hong et al. [6] improves the routing to take advantage of node heterogeneity, yet they assume a static network. In many scenarios, the nodes in a DHT may be frequently running other tasks, and thus their capacities are changing in real time. The protocols in [5, 6] do not scale well to real-time application.

Specific to the Kademlia DHT, there have also been attempts to address the heterogeneity of nodes [1, 9], but these approaches do not generalize to a dynamic environment. Kadobayashi [8] proposes an approach similar to ours: assign a weight to each node to represent capacity. The weight can be sent as part of the communication protocol and thus this approach can address the dynamic capacity. However, their method and experiments have two limitations. First, the selection of k neighbors is the same as the original Kademlia algorithm, and the weight is only taken into account in the subsequent α selection problem. Compared to ours, their protocol may miss some optimal routes because they are dis-

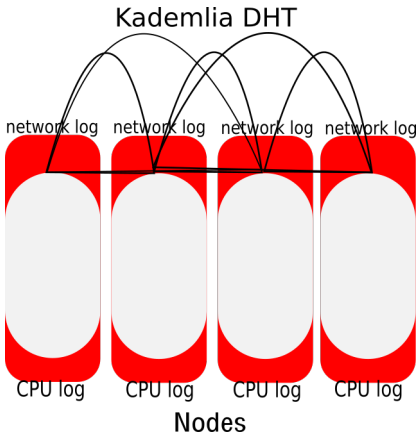

Figure 1: Graphical Representation of System

carded in the first step. Second, in their experiments, weights are of discrete values, which limits the expressivity.

Jiang et al's work at Huawei [7] is similar to our project in that it relies on using additional information stored in finger tables which can be updated efficiently in real-time. However, their system focuses on choosing backup resources in a multicast group. In contrast, the primary objective of our modification is finding a "better" path to a given resource rather than finding other versions of resources. However, ideas and lessons from their work, such as maintaining *business level* information to avoid busy nodes, can be incorporated to our work.

3 Solutions

Our solution to the issue is to introduce 1) a logging framework, and 2) an alternate metric taking into account performance information. A graphical representation of the system can be found in Figure 1.

3.1 Alternate Metric

To find out the alternate metric, we use a weighted average of the XOR distance, the alternate idle CPU power and the idle memory capacity. The standard Kademlia model takes into account solely the XOR distance between the hashed two nodes. By taking the weighted average of several performance metrics, we hope to show that the additional information about resource usage can improve the routing performance.

In more mathematical language, we have that our new metric is

$$d(x, y) = w_1 * k + w_2 * C + w_3 * M + w_4 * N \quad (1)$$

In the above equation, we have that k is the XOR key distance, C is the CPU capacity, M is the memory capacity and N is the network capacity.

The reason for the choice of a weighted average is that it is the simplest function of the parameters. We considered other more complex functions, such as those that take into account the standard deviation and other statistical methods, but we reasoned that we wanted the simplest possible test as a proof of concept, which was the use of a weighted average.

The weights for the weighted average was hard-coded by hand, due to time constraints. At first we gave equal weighting to all the parameters. Nonetheless, this is an easily adjustable parameter that leaves more opportunity for future work. It is also possible for the system to learn and adaptively change these parameters.

3.2 Logging Framework

The logging framework consists of two different parts: local logs being written to by process and a server-client system that allows any node to ping another node for system information. The goal of the framework is the attempt to capture the key statistics of a machine. This it does by capturing the output of Linux commands *nethogs*, *mpstat* and *uptime*. Each machine keeps track of its own logs, which is periodically updated to keep track of the last five logs. The reason for this selection of the last five logs is that they provide an adequate picture of what is happening in the system in real-time.

Linux Utility	Information	Units
nethogs	send bandwidth	KB/sec
nethogs	recv bandwidth	KB/sec
uptime	load average (past 1 minutes)	percentage
uptime	load average (past 5 minutes)	percentage
uptime	load average (past 15 minutes)	percentage
mpstat	user cpu usage	percentage
mpstat	system cpu usage	percentage
mpstat	idle cpu usage	percentage

Table 1: System Information Logged

The client-server system consists of a server process running on each node, waiting for pings, and the client which sends requests for system information on a need-to-know basis. The implementation uses TCP/IP, and we have developed a simple protocol for the sending of messages that begins with the client sending a simple hello message and the server sending a message consisting of all the requested system information, and ending with the client sending an acknowledgement message. The server-side of the system runs non-stop on each node. It is called by the main as a forked process, and it is reasonably robust against multiple requests, since the pinging sent by the client and the response by the server is very fast. The client, however, is also activated as a forked process when information is required. This allows the information sent to the client to be fresh, which is important in our system that wants to take into account real-time performance and resource availability.

3.3 Adaptive Routing

Having introduced the distance metric and the logging framework, we now present the adaptive routing protocol in Algorithm 1. Our protocol is built on top of the internal routing algorithm in the DHT. Both Chord and Kademlia use an iterative query to locate the destination. At each iteration, the DHT selects some candidates, from which the next-hop node is chosen. We insert our distance metric here. To adapt to real-time capacity, we leverage the logging framework to obtain the load information from each of the candidates.

Algorithm 1 Adaptive Routing in DHT

```
1: while target not found do
2:     DHT Routing selects some candidates
3:     Query candidates for load information
4:     Calculate distance metric
5:     Route to the candidate with smallest distance
6: end while
```

3.4 Maintaining the Convergence Guarantee

One question that should be addressed whether the convergence guarantee still holds with the new routing algorithm in place of the old one. Most, if not all, DHTs have a mathematically proven guarantee showing that any search request for an id in the key space will terminate in finite steps. Chord and Kademlia are examples, and they rely on the mathematical properties of the id keyspace and the distance metric they employ. The distance metrics can be the literal distance in the keyspace (Chord), or the XOR value of two binary ids. Thus, it is natural to ask if the new routing algorithm has this property. Although we do not provide a universal guarantee of convergence for all DHTs, we show that exploiting the proper-

ties of the structures or algorithms employed in Chord and Kademlia, we were able to preserve the convergence guarantee for the two DHTs. Chord's convergence is implemented and tested in our simulation program.

One key characteristic of Chord's routing algorithm is that no the node approaches the target from a clockwise direction. Therefore, to maintain the convergence guarantee with our new distance metric, we need only compare the nodes that are on the arc clockwise from the current node to the destination, and ignore the rest.

Likewise, Kademlia's routing protocol can be abstracted as executing Chord's routing in parallel through the usage of k-buckets, which is maintained to choose the k "best" nodes with respect to the metric function. To ensure convergence in finite steps, we propose that we reserve 1 bucket to choose according to the old distance metric, while the rest $k-1$ buckets are filled using the new function. However, we did not include this in our experiments.

4 Experiment & Results

We have conducted two different experiments as a proof-of-concept. The first is a simulation that simulates environment loads and a toy version of our model and the other is a realistic experiment conducted on eight virtual machine nodes.

4.1 Virtual Machine Experiment

Our implementation for the virtual machine experiment is based on OpenDHT, an open-source implementation of the Kademlia distributed hash table. OpenDHT is written in C++11, and we built on top of the open-source project. In total, we contributed around 2.5k lines of code on top of the original implementation. The logging system was implemented from scratch, and the testing framework that we built on top of OpenDHT was novel.

The setup for this experiment is the use of eight virtual nodes. Each virtual node has its own unique IP, which is what it uses to communicate to other nodes and to be uniquely identified by that address. It also has its own local memory and its own logging system.

We compare our implementation of the adaptive DHT and distance metric to the original Kademlia XOR. The main metric that we are using to judge performance is that of put and retrieve speed (in seconds). We also consider three different environments for the tests. An environment can either be a *low-computation regime*, a *high-computation regime*, a *low-bandwith regime*, and a *high-bandwidth regime*. To create the disparities in such regimes, we artificially create workload on the nodes, by using the Linux utility *stress*.

Our main results can be found in the Table 2.

4.2 Timing Out on `Get`

The results for `put` can be found in Table 3. We notice that the unaltered algorithm dominates that of the resource-aware model. The likely reason for this is that it pings for system information more than once, while across the entire network. The latency from such pings adds up rapidly to lead to very expensive put and get operations.

Based on our implementation, there is sometimes something wrong with the get operation. This could either be on a theoretical or an implementation problem. When we replaced the XOR with our weighted average version, get seemed to stop working perfectly. On approximately 5% of queries, the request would time out. There is no longer the assurance of perfectness of the result.

This curious phenomenon also did not occur on our two node experiments, which shows that it might be due to the complexity of the finger tables. We were unfortunately unable to exactly to pinpoint the exact source of the error, which could be either a theoretical error or a problem with our implementation.

Based on this we do not include the outliers in the averages for the get operation, since they

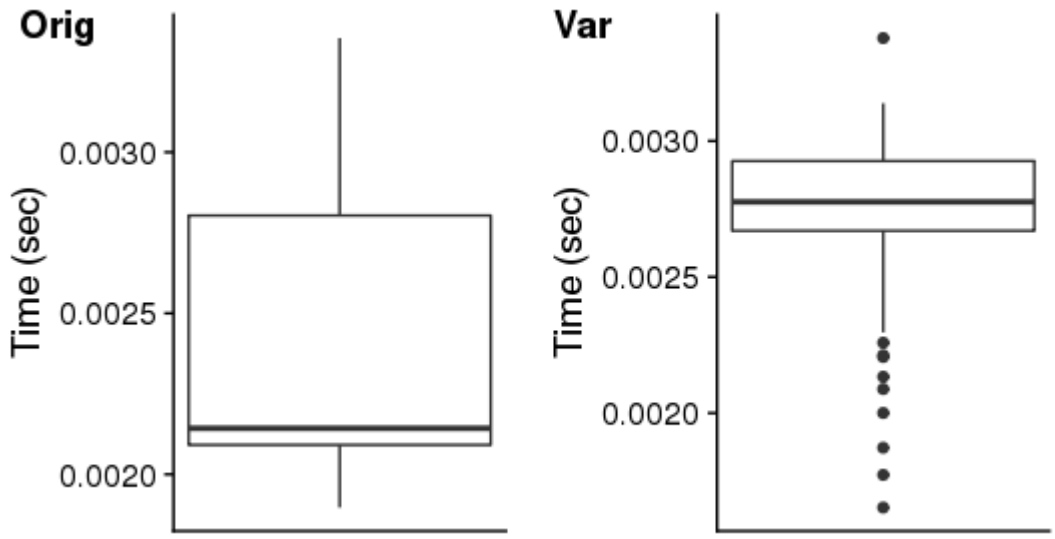

Figure 2: Comparison of the Put Low Bandwidth Case

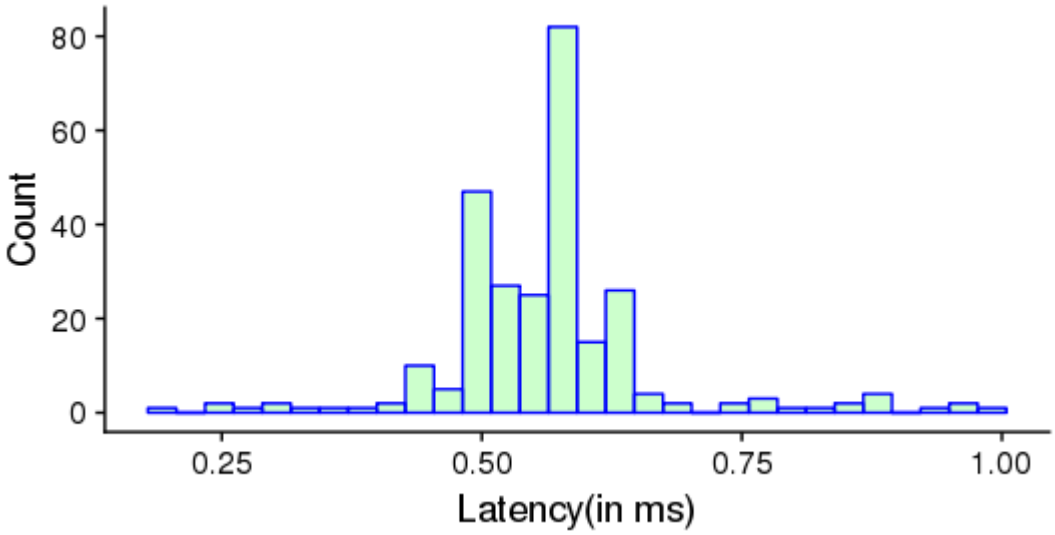

Figure 3: Distribution of Latencies

Environment	Normal	Resource-Aware
Low Bandwidth	0.00103	0.0027
High Bandwidth	0.00215	0.028
Low Computation	0.0010	0.0012
High Computation	0.00215	0.022

Table 2: Results of Put Experiments (All 150 Trials)

Statistic	Normal	Resource-Aware
Mean	0.0023	0.0933
1st Quartile	0.0018	0.079
Median	0.0020	0.096
3rd Quartile	0.0023	0.117

Table 3: Results of Get Experiments (100 Trials & High Bandwidth)

would unfairly skew the data.

4.3 Path Length and k in Simulation

Due to the small size and the limited nature of the available network (8 nodes running on VMs), we also ran experiments on a simulated overlay network.

For the experiment, we randomly created 1000 nodes with random unique ids in 2^{100}-sized keyspace, and initialized them with random resource states. Then, we generated 100 random unique ids to be put into the nodes, and sampled with replacement 100 ids from the put lists.

However, it is reasonable to expect that assigning a simulated time value for a hop from nodeA to nodeB in the simulation will not only be unrealistic to practical environment, but also overfitted to the new routing algorithm we test. Therefore, we instead measured the average length of routing paths, i.e. the number of nodes a routing visits in the course of locating the destination.

In Figure 4, *path1* represents the average length of path for the original algorithm, while *path2* that of our algorithm. Note that the x-axis represents the number of k, the size of the finger table, in other words, the number of neighbors

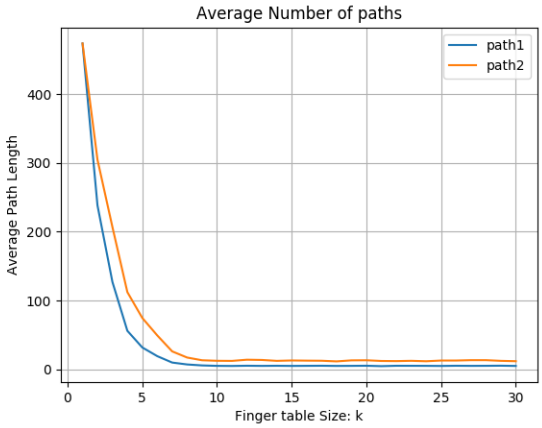

Figure 4: Finger Table Size and Path Length

whose addresses a node is aware of.

Looking at the graph, it is not surprising that *path1* is always below *path2*, since the original routing algorithm by definition takes the shortest route in terms of keyspace distance. However, it is notable that the discrepancy between *path1* and *path2* is largest in the lower range $(0 - 10)$ while both similarly plateau to very low levels as k becomes large.

The decreasing trend itself is not surprising, since larger finger tables means larger "hoppinh". However, it is plausible that the decreasing discrepancy of path length with large ks suggests that if a network is assumed to have nodes which act as severe bottlenecks, the resource-aware algorithm might result in significant gain in terms of speed, given that the "go-around" behavior only adds a few steps compared to the original routing scheme.

5 Conclusions

The conclusions that we draw is that 1) the logging system and the pinging system makes the system slower than the original Kademlia implementation, 2) that the environments rightly affect the performance of put and get, 3) that there are some issues associated with deviating from a theoretically correct model and 4) that there is much opportunity for further work.

5.1 Comparison to Normal

As can be seen in the side-by-side boxplots in Figure 1, we can notice the discrepancy between the original implementation and our implementation. The mean time for the original Kademlia operations are much faster than that for our variant. We hypothesize that the difference is largely due to the cost required to ping the system information, which adds latency to the put operation.

To see that this is a plausible explanation for the slowdown of our model, we refer to Figure 2. This is the plot of the distribution of the latency between two nodes on our system. We can see that the values of the latency between two nodes are distributed around 0.60 ms, which could account for almost one half of the difference in times between the original Kademlia and our variant. We therefore believe that we have ample evidence to claim that what we thought was one selling point of our system, the ability to query other systems was in fact a cause for the lack of scalability and performance.

5.2 Effects of Environments

As one can see in Table 2, the increase in environment load leads to an increase in the time. However, as one can see from the comparison with the normal Kademlia model.

5.3 The Issue with `Get`

When Kademlia was originally published, most of the paper was focused on the theoretical correctness of the algorithm. The parts that needed to work, such as the searching and the finger table were shown to be correct. However, get in our implementation does not seem to work perfectly. The possibility is that the correctness proof does not extend to our modified operation or that proofs of correctness are still crucially important, or the possibility that the modification of the code had a negative interaction of another part of the code. We found this disheartening, nonetheless it shows that a very interesting null result. This issue only seemed to appear when in the final testing phase when we increased the number of nodes, which again is a very interesting null result.

One possible explanation that we have for the issue is that at compilation time for the program, there is no way that the system can predict which nodes are going to be most overused. In the original Kademlia, however, the knowledge of the function is needed to place the nodes into bins in the first place. But we are still undecided on the exact source of the error.

6 Future Work

6.1 More realistic testing environment

A glaring limitation of our experiment in this work that the testing involves a very small number (8) of machines located in a single local area network (Swarthmore College). Thus, the result of the test cannot be easily generalized to a more general case, because typically many DHTs involves thousands or more nodes, many of them distributed over several networks and physically removed from each other by unlimited distance. To remedy this, future studies would have to obtain help from a large institute with access to machine, or recruit multiple volunteers in charge of individual nodes.

6.2 "Bootstrapping" the Simulation with real data

Alternative way to test a large overlay network is to simulate multiple nodes. However, as outlined in our experiment section, it is difficult to imitate a large, realistic network without "overfitting" the simulation parameters in the sense that we are adjusting the simulated network to behave in a way that our modified algorithm works well.

Another idea we developed but did not manage to test in the scope of this work is simulating a large network ($\tilde{1}000$ or mode nodes) from a data set generated by a real, but smaller (8, for example) network. Specifically, we propose the latency of a ping from a node A to another node B as a proxy for measuring the latency of inter node-communication in the process of DHT routing. Then, data points in the form of *(nodeA uniq id and states, nodeB id resource states, ping latency)* can be used to model the states and the routing time between nodes in a simulated network.

6.3 "Better" decision function

Because we used a simple weighted average to be used with our XOR function, there are possibilities of using more complex functions. One fruitful way to think of the XOR function is as a way of measuring closeness or distance as mapped to True or False. Therefore, methods that lead to classifying two nodes as close or far can be very effective. There is the possibility of using machine learning methods to predict closeness based on the system information.

6.3.1 A Grid Search of Algorithm Parameters

Moreover, the choice of our parameters, i.e the weights, has largely been arbitrarily made, although a few iterations of trial-and-error adjusting has been made. This means that although in some cases our modified algorithm showed improved performance, there might be a set of more *optimal* parameters. In order to search for this parameters, we propose a grid search of parameters over a data set gathered from simulated, "bootstrapped" networks mentioned above, but leave the challenge for future work.

6.3.2 Caching

Another possibility that helps with the latency problem of communicating using the client-server model is the use of caching. We hypothesize that caching node information and updating as necessary will save a lot of time. This saves on the network latency, which we have argued for the main cause of the slowdown of our variant compared to the original.

7 Meta-discussion

The hardest parts of the project were working with an already well-established codebase, and configuring multiple machines over a network.

7.1 Working with Large Codebase

The difficulty of working with a large and well-established codebase like OpenDHT is that there is it requires familiarity and understanding of the organization of the programs, and knowledge of language-specific syntax. To add to this problem, OpenDHT did not come with a user-friendly documentation, nor were the source code well commented or explained thoroughly. If we had to do this project over again, we would like to have implemented everything from scratch, for a large portion of the time spent for this project was invested in obtaining a functional comprehension of the codebase.

Abstraction is a programming concept where the details of the lower-level implementation are hidden to the user. This is in contrast to the concept of end-to-end system principles which gives the freedom and the ability to manipulate the code to the user. What we have learned is that there is a certain downside to abstraction when there is a need to break it. For example, in our

use of OpenDHT, whenever we had to change a fundamental part of the core of the system, it took an enormous amount of time and effort. This is because we are in effect breaking the abstraction to get to the core of the code.

7.2 Configuring Nodes/Lack of Simple Networking Tools and Framework

When dealing with the multiple nodes as in distributed computing, we rapidly found that our favorite tools from the sequential world were not useful at all. For example, something as intuitive as *bash* scripting in the sequential world was made exponentially more difficult in the distributed world, as did other tools such as *valgrind*.

Another type of difficulty was manipulating the eight nodes that we were given at once. When we had to update and download the packages, it was a nightmare to have to do that over the nodes at once. Again, problems easy in the sequential world such as running a script on a node became a complicated mess of forking and communication.

References

[1] Andreas Binzenhöfer and Holger Schnabel. Improving the performance and robustness of kademlia-based overlay networks. In *Kommunikation in Verteilten Systemen (KiVS)*, pages 15–26. Springer, 2007.

[2] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, January 2002.

[3] Rubén Cuevas, Manuel Uruena, and Albert Banchs. Routing fairness in chord: analysis and enhancement. In *INFOCOM 2009, IEEE*, pages 1449–1457. IEEE, 2009.

[4] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with coral. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 18–18, Berkeley, CA, USA, 2004. USENIX Association.

[5] P Brighten Godfrey and Ion Stoica. Heterogeneity and load balance in distributed hash tables. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 596–606. IEEE, 2005.

[6] Feng Hong, Minglu Li, Xinda Lu, Jiadi Yu, Yi Wang, and Ying Li. Hp-chord: A peer-to-peer overlay to achieve better routing efficiency by exploiting heterogeneity and proximity. In *International Conference on Grid and Cooperative Computing*, pages 626–633. Springer, 2004.

[7] Haifeng Jiang, Feng Li, Xingfeng Jiang, and Lei Han. Packet routing method, system, device and method, system for selecting backup resource.

[8] Youki Kadobayashi. Achieving heterogeneity and fairness in kademlia. In *Applications and the Internet Workshops, 2004. SAINT 2004 Workshops. 2004 International Symposium on*, pages 546–551. IEEE, 2004.

[9] Imre Kelényi and Jukka K Nurminen. Optimizing energy consumption of mobile nodes in heterogeneous kademlia-based distributed hash tables. In *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST'08. The Second International Conference on*, pages 70–75. IEEE, 2008.

[10] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop*

on Peer-to-Peer Systems, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[11] Marti A. Motoyama and George Varghese. Crosstalk: Scalably interconnecting instant messaging networks. In *Proceedings of the 2Nd ACM Workshop on Online Social Networks*, WOSN '09, pages 61–68, New York, NY, USA, 2009. ACM.

[12] Johan Pouwelse, PawełGarbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Proceedings of the 4th International Conference on Peer-to-Peer Systems*, IPTPS'05, pages 205–216, Berlin, Heidelberg, 2005. Springer-Verlag.

[13] Venugopalan Ramasubramanian and Emin Gün Sirer. The design and implementation of a next generation name service for the internet. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 331–342, New York, NY, USA, 2004. ACM.

[14] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, 2001. ACM.

[15] Sylvia Ratnasamy, Ion Stoica, and Scott Shenker. Routing algorithms for dhts: Some open questions. In *International Workshop on Peer-to-Peer Systems*, pages 45–52. Springer, 2002.

[16] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, UK, 2001. Springer-Verlag.

[17] Stefan Saroiu, P Krishna Gummadi, and Steven D Gribble. Measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking 2002*, volume 4673, pages 156–171. International Society for Optics and Photonics, 2001.

[18] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[19] Hao Zhang, Yonggang Wen, Haiyong Xie, and Nenghai Yu. A survey on distributed hash table (dht): Theory, platforms, and applications. 2013.

[20] Hui Zhang, Ashish Goel, and Ramesh Govindan. Incrementally improving lookup latency in distributed hash table systems. In *ACM SIGMETRICS Performance Evaluation Review*, volume 31, pages 114–125. ACM, 2003.

[21] B. Y. Zhao, Ling Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE J.Sel. A. Commun.*, 22(1):41–53, September 2006.