Proposal

Andrew Rosen

September 29, 2014

Contents

Chapter 1

Hypotheses

1.1 DHTs are better for distributed computing under many circumstances

Distributed Hash Tables (DHTs) are traditionally used as the backbone of Peer-to-Peer (P2P) file-sharing applications and research has largely remained in this area. However, they are seeing increasing use in other applications:

- Using a DHT as the name resolution layer of a large distributed database [1].
- Distributed machine learning [2].
- Cloud Provisioning (P2P cloud Provisioning).

[1]

1.1.1 Really Cool Qualities

Many DHTs are built upon the assumption that they will be used in some kind of P2P application. This means a successful application must be deployable on very varied network sizes. The network must be able to handle members joining and leaving arbitrarily. The application must be agnostic towards hardware. The application must be decentralized and split the burden relatively equally among its members.

[1] Many papers use different terms to describe congruent elements of DHTs, as some terms may make sense I shall endeavor to add to confusion by using the following unified terminology:

Peerlist - The set of all peers that a node knows about. This is sometimes referred to as the *routing table*, but certain DHTs [3] [4] overload the terminology.

Neighbors - The subset of peers that are "closest/adjacent" to the node in the keyspace, according to the DHT's metric. In a 1-dimensional ring, such a Chord [5], this is the node's *predecessor* and *successor*.

Fingers - The subset of the peerlist that the node is not adjacent to. These are sometimes referred to as long-hops or shortcuts.

Root Node - The node responsible for a particular key.

This leads to DHTs have very specific properties. While these properties may be individually enumerated, they are greatly intertwined.

Robustness and Fault-Tolerance

One of the most important assumptions of DHTs is that they are deployed on a non-static network. DHTs need to be built to account for a high level what is called *churn*. Churn refers to the disruption of routing caused by the constant joining and leaving of nodes. This is mitigated by a few factors.

First, the network is decentralized, with no single node acting as a single point of failure. This is accomplished by each node in the routing table having a small portion of the both the routing table and the information stored on the DHT (see the Load Balancing property below).

Second is that each DHT has an inexpensive maintenance processes that mitigates the damage caused by churn. DHTs often integrate a backup process into their protocols so that when a node goes down, one of the neighbors can immediately assume responsibility. The join process also causes disruption to the network, as affected nodes have adjust their peerlists to accommodating the joiner.

The last property is that the hash algorithm used to distribute content evenly across the network(again see load balancing) also distributes nodes evenly across the DHT. This means that nodes in the same geographic region occupy vastly different positions in the keyspace. If an entire geographic region is affected by a network outage, this damage is spread evenly across the DHT, which can be handled.

This property is the most important, as it deals with failure of entire sections of the network, rather than a single node. Recent research in using DHTs for High End Computing [6] shows what can happen if we remove this assumption by placing the network that is almost completely static.

Load Balancing

All Distributed Hash Tables use some kind of consistent hashing algorithm to associate nodes and file identifiers with keys. These keys are generated by passing the identifiers into a hash function, typically SHA-160. The chosen hash function is typically large enough to avoid hash collisions and generates keys in a uniform manner. The result of this is that as more nodes join the network, the distribution of nodes in the keyspace becomes more uniform, as does the distribution of files.

However, because this is a random process, it is highly unlikely that each node will be spread evenly throughout the network. This appears to be a weakness, but can be turned into an advantage in heterogenous systems by using *virtual nodes* [7] [8]. When a node joins the network, it joins not at one position, but multiple virtual positions in the network [8]. Using virtual nodes allows load-balance optimization in a heterogeneous network; more powerful machines can create more virtual nodes and handle more of the overall responsibility in the network. Strategies are discussed further in the Heterogeneity subsection.

Scalability

In order to maintain scalability, a DHT has to ensure that as the network grows larger:

- Churn does not have a disproportionate overhead.
- Lookup request speeds (usually measured in hops) grow by a much smaller amount, possibly not at all.

Using consistent hashing allows the network to scale up incrementally, adding one node at a time [8]. In addition, each join operation has minimal impact on the network, since a node affects only its immediate neighbors on a join operation.

Similarly, the only nodes that need to react to a node leaving are immediately. This is almost instantanius if the network is using backups. Other nodes can be notified of the missing node passively through maintenance.

There have been multiple proposed strategies for tackling scalability, and it is these strategies which play the greatest role in driving the variety of DHT architectures. Each DHT must strikes a balance between memory cost of the peerlist and lookup time. The vast majority of DHTs choose a logartihmic sized routing table Table 1.1 lists various DHTs and methods of balancing cost.

DHT	Routing Table Size	Lookup Time	Join/Leave	Comments
Chord [5], Kademlia [9]	$O(\log n)$	$O(\log n)$		This is where most DHTs fall
CAN [10]	$\Omega(2d)$	$O(n^{\frac{1}{d}})$, average $\frac{d}{4} \cdot n^{\frac{1}{d}}$	Affects $O(d)$ nodes	d is the number of dimensions
Plaxton-based DHTs, Pastry [4], Tapestry [3]				
ZHT [6]	$O(n)$	$O(1)$		Assumes an extremely low churn

Table 1.1: The different ratios and their associated DHTs

Heterogeneity

The mechanics behind load balancing assumes nothing about the nature of the hardware it is running on. The applications that run DHTs, on the otherhand, implicitly assume that that the machines composing the network (and running the application) are heterogeneous.

Can't automatically assign work natively (or can you: end project automatic load balancing DHT?) Can assign work manually with virtual nodes

1.1.2 The Takeaway

- DHTs are extremely good if your problem is embarrassingly parallel

- DHTs are agnostic in terms of what hardware it's running on.
-

The consequences of the Properties

So what are the consequences of these properties?

- DHTs can use constistant hashing supplemented by virtual nodes to efficiently load-balance.
- DHTs are highly resilient to damage and can handle abnormally high rates of disruption. This is extremely desirable in a DHT
- X is a desirable property in a network for distributed computing by
-

The consequences of the Properties

1.2 Different or subproblem: Certain DHTs are better at one application than another due to differences

1.2.1 Design Differences Impacts

1.2.2 Geometries

1.2.3 Routing Table Construction

1.2.4 Implementation Differences Impacts

Recursive or iterative seek

1.2.5 MANETS

Chapter 2

Justification and Why I Think It's Cool

2.1 Why DHTs for distributed computing?

[11] - Between congestion, cost of join/leaves, and lookup time there are tradeoffs. Optimizing for two can be done but has bad cost. For example, a balanced binary tree has congestion at root.

2.1.1 DHTs well understood

2.1.2 DHTs are Highly used for their intended purposed

Bittorrent, WoW

2.2 DHTs are being effectively leveraged for other things besides file sharing already

PaaS

Load Balancing in the cloud

Computing is a natural extension

Chapter 3

Possible Experiments and Applications

3.1 MapReduce

Both MapReduce [12] and DHTs are based on operating over key/value pairs, albeit over different contexts.

MapReduce can utilize a DHT to create a more abstract MapReduce frameworkk

3.1.1 Current MapReduce DHT/P2P combos

Example A: slightly decentralized with coordinating nodes. Also limap

ChordReduce

The idea behind ChordReduce is

ChordReduce is completely decentralized, no node is more vulnerable than any other.

3.1.2 Experiment Description: Comparison of MapReduce paradigm on different DHTs

This may be enough on its own.

- Implement CAN, Pastry, Chord, Kademlia, VHash and ZHT/similar
  - This covers different geometries with different base paremeters
  - This also necessitates the creation of an extensible DHT framework, something I haven't found yet and may not have been done before (most people implement their DHT or use someone else)
- Compare results with each other and MapReduce

3.2 High End Computing

3.2.1 Metadata Management

3.2.2 Robustness

3.2.3 Experiment Description:

3.3 Graph Processing on a DHT

Lookup Graphlab

3.3.1 Embedding

3.3.2 Experiment Description:

3.3.3 Distribute the work for solving a graph on a DHT

3.3.4 Comparison to well established or state of the art methods

3.4 Machine Learning Problems on A DHT

Bayesian Learning

3.4.1 Experiment Description:

3.5 DHTs as a volunteer Platform

3.5.1 Experiment Description

Implement and compare to Boinc.

3.6 Resources

3.6.1 Planetlab

3.6.2 Local Cluster

Chapter 4

DHT Background

Bibliography

[1] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid computingâĂŤwhere {HPC} meets grid and cloud computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 440 – 453, 2011.

[2] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning,"

[3] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, pp. 41–53, 2004.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*, pp. 329–350, Springer, 2001.

[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, pp. 149–160, August 2001.

[6] T. Li, X. Zhou, K. Brandstatter, D. Zhao, K. Wang, A. Rajendran, Z. Zhang, and I. Raicu, "Zht: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 775–787, IEEE, 2013.

[7] P. B. Godfrey and I. 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*, vol. 1, pp. 596–606, IEEE, 2005.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS Operating Systems Review*, vol. 41, pp. 205–220, ACM, 2007.

[9] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*, pp. 53–65, Springer, 2002.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," 2001.

[11] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," 2001.

[12] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.