Quantifying the Tradeoffs Between Distributed Hash Tables and Distributed Directories

Hakim Weatherspoon, Byung-Gon Chun, Jeremy Stribling[1], John D. Kubiatowicz
University of California, Berkeley
{hweather, bgchun, kubitron}@cs.berkeley.edu and [1]strib@csail.mit.edu

Abstract

We explore the possibility of achieving robust and efficient peer-to-peer storage by differentiating between reliable and unreliable nodes. Central to our technique is the use of a *distributed directory* (DD) to provide a level of indirection, thereby permitting the system to exploit heterogeneity. We compare and contrast the DD approach to a *distributed hash table* (DHT) when constructing reliable storage systems from components of varying reliability. By comparing these techniques, system architects can design systems that are more efficient and scalable. For instance, we conclude with a configuration that is durable, available, and scalable with a Gnutella-like distribution of node reliability. Our results show that a DD uses upto two orders of magnitude less bandwidth per node than a DHT to maintain data.

1 Introduction

Maintaining state in a structured P2P overlay network using minimal resources is a non-trivial problem (e.g., less than 15% of the bandwidth capacity of each node). While such overlays have been shown to handle dynamic node membership in both theory and practice [8, 9, 11, 12, 14, 17], the bandwidth required to do so can be quite expensive. Several proposed storage applications [3, 5] leverage the fact that structured P2P overlays handle churn efficiently and gracefully by using the overlay maintenance mechanisms directly (i.e., the overlay remains connected under constant or transient change [10]). However, Blake and Rodrigues [2] showed that tightly coupling the storage and networking in this manner is prohibitively expensive. The bandwidth required to maintain data availability and durability in a dynamic network exceeds the bandwidth capacity, given that the disks of the participants are well-utilized. In other words, participants can only contribute an insignificant amount of disk space to keep the maintenance bandwidth below link capacities.

The fundamental problem with combining networking and storage is that the two tasks require separate policies for efficient use and maintenance. In this paper, we show that by adding a level of indirection at the storage level, the separate layers can be maintained efficiently with different policies using similar mechanisms. The contribution of this paper is quantifying the cost and benefit of a layer of indirection and analyzing the tradeoffs, an analysis not yet seen in the body of P2P literature. Given these tradeoffs, we show performance comparisons between coupled and decoupled storage designs. Our final goal is an efficient *self-organizing* option.

Section 2 begins by comparing the costs and benefits between *distributed hash tables* (DHTs), in which networking and storage are united, and *distributed directories* (DDs), in which they are separated. After describing our analytical model in Section 3, we provide a quantitative evaluation and comparison of the two solutions in Section 4. We discuss these results and their implications in Section 5.

2 Design Comparisons

Storage solutions, such as CFS [3] and PAST [5] have proposed using DHT mechanisms and policies directly, and as a result are both simple and elegant. However, such solutions assume servers have independent and identically distributed availability and failure distributions, use replication or other forms of redundancy to compensate for server faults and failures, and argue that random placement of replicas are sufficient to maintain desired levels of data availability. Unfortunately, the bandwidth required to maintain the redundancy is the bottleneck [2]. Furthermore, selective placement of data may be desirable for some systems [7, 16]. In this section we compare and contrast the design points for DHT-based and DD-based storage solutions.

2.1 Distributed Hash Tables

The distributed hash table (DHT) abstraction [4] maps a given key in the node/object identifier space to a particular node in the overlay. An application can insert data into the DHT, which places that data on the node responsible for the key of the data; the data is retrieved by using the DHT to find the responsible node and asking it for the data. The DHT maintains this mapping between key and node consistently, even as nodes join and leave the network.

DHT Pros: The DHT abstraction is an elegant solution for maintaining the storage layer because the mapping function of the DHT inherently decides who owns the mapping of a key to an object, when to remap that responsibility, and where the responsible

node is located. This shields the storage layer from making similar decisions.

DHT Cons: The problem with using the DHT abstraction directly for the storage layer is that it provides no control over data placement, causing maintenance bandwidth to become prohibitively expensive under churn. Tightly coupling the storage layer with the networking layer wastes bandwidth by automatically transferring data each time the key for that data is remapped; this transfer may take place over the wide area to a node that may only live for a short period of time. It also may remap responsibility for data away from a node experiencing a short transient failure, potentially wasting wide area bandwidth.

2.2 Distributed Directories

A distributed directory (DD) decouples the storage layer from the underlying transient network. At its core, a DD is a level of indirection–utilizing pointers within the network to achieve flexibility in replica placement. Since DD functionality does not require locality, it can be provided by any DHT that can reliably store small pointers to objects. DD functionality is also provided by Decentralized Object Location and Routing (DOLR) layers [4]. The DD approach to storing data is a hybrid technique: it utilizes the peer-to-peer system to maintain pointers and more sophisticated methods to maintain data.

DD Pros: Decoupling the storage layer from the networking layer insulates the data from the transient network, saving wide area bandwidth. Also, the separation allows the placement of data to be biased towards more available and reliable nodes, increasing data availability and durability. Finally, since a DHT still identifies a *root* node responsible for the pointer of each object, the repair of an object can be *triggered* after a specified threshold has been reached, further insulating the data layer.

DD Cons: The extra level of indirection must be maintained. The pointers need to be replicated to prevent *memory leaks* (i.e. a live node cannot locate data that exists). Another issue is outdated pointers; namely, *dangling pointers* that no longer point to the correct location for data. Additionally, system designers need to be careful that the aggregate storage due to pointers does not increase the maintenance bandwidth; that is, is some small percentage of the total storage. Furthermore, the storage layer becomes more complex because it requires an independent data placement and replication mechanism.

The remainder of this paper analyzes and compares the cost of DHT and DD storage systems. By understanding the tradeoffs, system architects can design systems that are more efficient and scalable.

Model	Description
model(1)	DHT spreads data evenly over all nodes
model(2)	DD spreads data over reliable nodes only and pointers over all nodes
model(2.a)	Reliable model(2) nodes. Stores all data and stores $\frac{N_R}{N}$ fraction of pointers.
model(2.b)	Unreliable model(2) nodes. Stores $\frac{N_U}{N}$ fraction of pointers (and no data).
model(3)	DD spreads data and pointers over reliable nodes only. (unreliable nodes are not used)
model(3.a)	Reliable model(3) nodes. Stores all data and pointers. (Similar to a DHT of reliable nodes only).
model(3.b)	Unreliable model(3) nodes. Not used (i.e. stores no data or pointers), $\frac{BW}{N_U} = 0$

Table 1: Storage Models. *Defines models used to compare different storage solutions, using various combinations of storing pointers and data on (un)reliable nodes.*

3 Model

The goal of this paper is to explore different configurations that allow the storage layer to be robust and bandwidth-efficient. By robust, we mean that data is durable and available with high probability. The storage layer also needs to be bandwidth-efficient in maintaining storage. Finally, the system needs to recognize and use new resources efficiently, as well as compensate for absent resources. This section describes the model we use to quantify these properties.

3.1 Background

The data maintenance bandwidth per node is approximated by the following equation derived by Blake and Rodrigues [2]:

$$\frac{BW}{N} = \frac{2kD}{Na} \cdot \frac{1}{T} \quad (1)$$

Where D is the amount of unique data in the system and k is the redundancy factor required to achieve a desired level of data availability. T is the average lifetime of a node in the system, and N is the total number of nodes. The number of nodes available to store data is reduced by the average node availability, a. Given a target data availability (i.e. $1 - \epsilon$, for some small ϵ), Blake and Rodrigues derive equations for the redundancy factor needed for replication and coding [1, 15] schemes (k_r and k_c respectively).

All values and comparisons throughout the rest of the paper will be based on extending Equation 1 to account for different data allocation and heterogeneous availability and lifetime.

3.2 Parameter Setting

We use three basic models to compare DHT and DD based storage solutions. The first model uses a DHT

to distribute data evenly over all nodes. The second model uses a DD to distribute indirection *pointers* evenly over all nodes and distributes the data only over the reliable nodes. The third model uses a DD to distribute the pointers and data only over the reliable nodes (similar to a DHT of only reliable nodes); unreliable nodes are not used for storage. We refer to the models throughout the rest of the paper as model(1), (2), and (3), respectively. Table 1 summarizes these models. The equations given for the data maintenance bandwidth per node, given N_R reliable nodes and N_U unreliable nodes (where $N_R + N_U = N$) are as follows:

Model (1)

$$\frac{BW}{N} = \frac{2k_r D}{N} \cdot \left\langle \frac{1}{aT} \right\rangle_N \quad (2)$$

Model (2)

$$\frac{BW}{N_R} = \frac{2k_r(D + k_p \cdot P\frac{N_R}{N})}{N_R} \cdot \left\langle \frac{1}{a_R T_R} \right\rangle_{N_R} \quad (3)$$

$$\frac{BW}{N_U} = \frac{2k_r \cdot k_p \cdot P\frac{N_U}{N}}{N_U} \cdot \left\langle \frac{1}{a_U T_U} \right\rangle_{N_U} \quad (4)$$

Model (3) (notice $\frac{BW}{N_U} = 0$)

$$\frac{BW}{N_R} = \frac{2k_r(D + k_p \cdot P)}{N_R} \cdot \left\langle \frac{1}{a_R T_R} \right\rangle_{N_R} \quad (5)$$

The term $\left\langle \frac{1}{aT} \right\rangle_N$ denotes the average of the inverse combination of availability and lifetime, for the set of N nodes. P is the amount of unique pointers in the system, corresponding to the amount of unique data D. The total storage for the DHT (model(1)) is $k_r D$, but for the DD it is $k_r(D + k_p P)$ (the pointers are replicated k_p times to prevent memory leaks). We defined the redundancy factor in terms of the replication factor k_r; for coding, we replace the replication factor k_r with the expansion factor k_c. Note that in the case of coding $k_r k_p P \Rightarrow k_c k_p m P$; that is for each unique object, $k_c m$ unique fragments are produced, and a pointer is needed for each fragment. The result is that pointers could cost more to store than the data itself, unless the ratio of data to pointers is greater than k_p (i.e. $\frac{k_r D}{k_r P} = \frac{D}{P} > k_p$ for replication and $\frac{k_c D}{k_c P} = \frac{D}{P} > k_p m$ for coding). The changes are summarized below:

$$\left\langle \frac{1}{aT} \right\rangle_N = \frac{1}{N} \cdot \sum_{i=1}^{N} \frac{1}{a_i T_i}$$

$$kD \Rightarrow k_r(D + k_p P) \text{ (for DD replication)}$$

$$kD \Rightarrow k_c(D + k_p m P) \text{ (for DD coding)}$$

We used the Overnet trace [1] to form a basis for comparing for models (1), (2), and (3). Table 2 summarizes the values we use for the comparisons[1].

[1]We do not extract T from the trace because T is only one week. The longest trace, Gummadi *et al* [6], is five months long; however, their passive trace only considers a node availabile when actively downloading an object. Instead, we show how maintenance bandwidth per node varies as the ratio $\frac{T_R}{T_U}$ varies.

Param	Value	Description
D	1TB	unique data
N	1469	total nodes
N_R	136	nodes with ≥ 0.7 availability (9.26% of N)
N_U	1333	nodes with < 0.7 availability (90.74% of N)
a	0.3	average availability of N
a_R	0.8	average availabilty of N_R
a_U	0.2	average availabilty of N_U
max T_R	256	max lifetime (in days) for reliable nodes.
max T_U	1	max lifetime (in days) for unreliable nodes.
$1 - \epsilon$	6 9s	data availability
k_r	10	replication factor to achieve data availability
k_c	2	coding factor to achieve data availability
k_p	10	pointer replication factor to avoid memory leaks
$m\ (= b)$	16	fragments required for reconstruct under coding

Table 2: Baseline Trace and Comparison Parameters. *The parameters N (N_R and N_U) and a (a_R and a_U) were extracted from the trace conducted by Bhagwan* et al *[1]. The rest of the parameters were derived from N and a, or chosen to be a "reasonable" setting.*

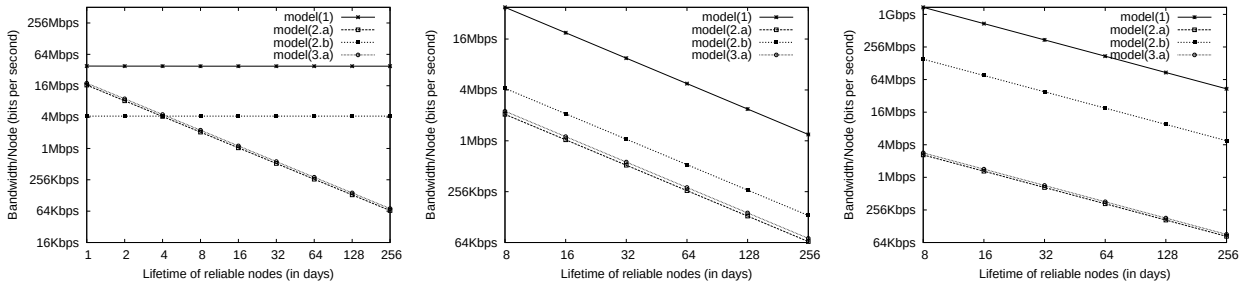
4 Quantitative Comparison

In this section, we compare four aspects of the different models, and their impact on data maintenance bandwidth per node. First, we vary the lifetime of the reliable nodes in relation to the lifetime of unreliable nodes. Second, we vary the data to pointer ratio, $\frac{D}{P}$. Third, we compare replication to coding. Finally, we explore the effect of varying the network size N. We use the architecture models summarized in Table 1 and the system parameter values summarized in Table 2 to compare DHT and DD storage systems. We only vary one parameter at a time indicated by the subsection heading.

4.1 Varying Lifetime, T

In Figure 1.a we hold T_U constant at 1 day, and vary T_R from 1 to 256 days. For model(1) (the standard DHT), the bandwidth used to maintain data availability in the face of unreliable nodes dominates the bandwidth cost and holds it steady at 32Mbps. For model(2), the reliable nodes bandwidth decreases as T_R is increased, but the unreliable nodes bandwidth stays constant at 4Mbps because pointers are being maintained across all nodes. For model(3), the reliable nodes maintenance bandwidth decreases as the lifetime increases. Since the unreliable nodes are not being used for data storage, model(3) is able to achieve a low per-node maintenance bandwidth of 64Kbps when the lifetime is 8 months.

For Figure 1.b we vary reliable node lifetimes along the x-axis and vary the unreliable node life-



(a) (b) (c)

Figure 1: Per-node maintenance bandwidth vs Lifetime. *Data/Ptr = 100 for (a), (b), and (c). (a) Per node maintenance bandwidth versus the lifetime ratio of the reliable and unreliable nodes, $\frac{T_R}{T_U}$. We increase $\frac{T_R}{T_U}$ by holding T_U constant at one day and increasing T_R. (b) $T_U = \frac{T_R}{8}$. The x-axis is T_R. (c) $\forall_i, T_i = a_i T_{max}$. The x-axis is T_{max}.*

times at a ratio of eight times less than the reliable nodes lifetime. The per-node maintenance bandwidth for all models decreases as lifetime increases; the models vary only by a constant difference in bandwidth. This constant difference is due to the constant difference in the combination of node availability, lifetime and the ratio of data to pointers.

For Figure 1.c we use a distribution for lifetimes among individual nodes. We vary lifetime in a similar manner conjectured by Gummadi *et al* [6]; that is, a node's lifetime is proportional to its availability (i.e. $T_i = a_i T_{max}$). As before there is a constant difference between the different per-node bandwidth maintenance costs, but it is dramatically increased by the difference in lifetime.

4.2 Varying Data to Pointer Ratio, $\frac{D}{P}$

For this comparison we consider only replication as we vary the data to pointer ratio in Figure 2.a In model(1), the bandwidth is constant at 32Mbps, due to the maintenance of data placed on unreliable nodes. For model(2.a), the bandwidth of the reliable nodes is fairly constant at 64Kbps; however, it is higher than 64Kbps when $\frac{D}{P} \leq k_p = 10$. For model(2.b), as we decrease the amount of pointers in the system (i.e. increase $\frac{D}{P}$), we decrease the amount of work each unreliable node is required to do. For model(3.a) reliable node maintenance is constantly low at 64Kbps (bandwidth per node is higher than 64Kbps when $\frac{D}{P} \leq 10$).

4.3 Replication vs. Coding, Varying $\frac{D}{P}$

In this section, we compare the maintenance bandwidth of replication versus coding. This analysis allows to to evaluate if more efficient coding techniques can actually be leveraged, or if the increased complexity of the infrastructure cancels out the redundancy gains made by coding techniques.

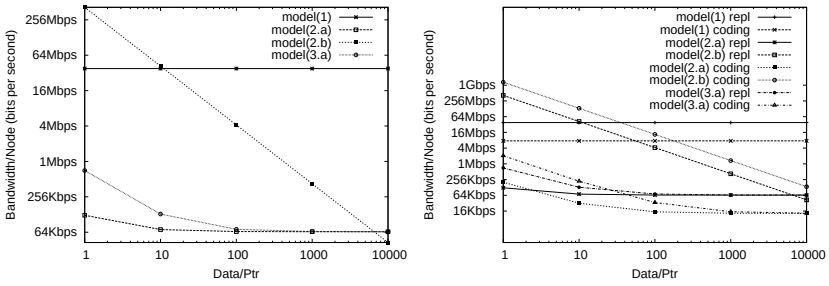
We compare both replication and coding schemes by varying the data to pointer ratio Figure 2.b. For model(1), the bandwidth per node is decreased by using coding techniques. The bandwidth per node is still dominated by the unreliable nodes, but coding reduces bandwidth from 32Mbps to 8Mbps. For model(2.a) the reliable nodes bandwidth is fairly constant and reduced from 64Kbps to 16Kbps by using coding techniques. The interesting point is that maintaining fragments is actually more expensive than maintaining replicas (in terms of bandwidth per node) when $\frac{D}{P} \leq k_p m = 10 \cdot 16 = 160$. For model(2.b), the DHT is maintaining the pointers. The DHT does not discriminate between reliable and unreliable nodes, so the unreliable nodes dominate the maintenance cost. But as we decrease the amount of pointers in the system (i.e. increase $\frac{D}{P}$), we decrease the amount of work the DHT is required to do. Another interesting point is that maintaining coding pointers is always more expensive than maintaining replication pointers by a constant factor m (i.e. $k_p < k_p m$). For model(3.a) reliable node maintenance is constantly low at 16Kbps when $\frac{D}{P} > 160$.

4.4 Varying Number of Node, N

In Figure 3, we show that the system aggregate storage scales linearly as the number of nodes increase. That is, bandwidth per node characteristics is constant because the storage per node is constant.

5 Discussion and Conclusion

DHTs do not take into account the suitability of a given peer for a specific task before explicitly or implicitly delegating that task to the peer[13]. In our analysis, differentiating among high- and low-availability nodes saves bandwidth. The savings increase as the gap widens, and if lifetimes are longer for highly available nodes (evident in Figures 1). Finally, storing pointers only works if the amount of data to pointer ratio is high (Figures 2).



(a) (b)

Figure 2: Per-node maintenance bandwidth vs Data/Ptr ratio. $T_R = 8$ *months and* $T_U = 1$ *day for both (a) and (b). (a) Per node maintenance bandwidth versus the Data/Ptr ratio for replication only. (b) Per node maintenance bandwidth versus the Data/Ptr ratio for both replication and coding.*

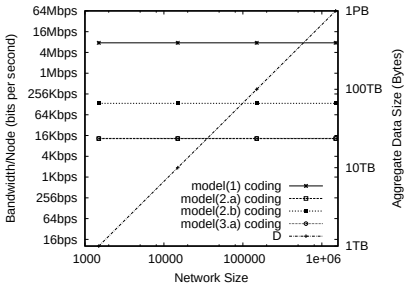


Figure 3: Per-node maintenance bandwidth vs Network Size. *We increase network size and aggregate storage and show that the maintenance bandwidth remains constant.*

By understanding the system dynamics and parameters, we can create a self-organizing option that is durable and available. One self-organizing option, is to use a DD. Figure 3 used the self-organizing mode to achieve six 9's of data availability. The DD configuration used a DHT of all nodes for pointers and reliable nodes for data, used coding, had an average of large objects, the average lifetime of the reliable nodes was eight months, and the reliable nodes were designated as having availability greater than or equal to 70%. Another non-self-organizing option for robust and bandwidth-efficient storage is to explicitly designate two rings [7], one reliable ring used for storage another unreliable ring not used at all.

References

[1] R. Bhagwan, S. Savage, et al. Understanding availability. In *Proc. of IPTPS*. February 2003.

[2] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. In *Proc. of HOTOS*. May 2003.

[3] F. Dabek, M. F. Kaashoek, et al. Wide-area cooperative storage with CFS. In *Proc. of ACM SOSP*. October 2001.

[4] F. Dabek, B. Zhao, et al. Towards a common API for structured P2P overlays. In *Proc. of IPTPS*. Feb. 2003.

[5] P. Druschel and A. Rowstron. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. of ACM SOSP*. 2001.

[6] K. P. Gummadi, R. J. Dunn, et al. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of ACM SOSP*. Oct. 2003.

[7] N. J. Harvey, M. B. Jones, et al. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of USITS*. March 2003.

[8] K. Hildrum, J. Kubiatowicz, et al. Distributed object location in a dynamic network. In *Proc. of ACM SPAA*, pp. 41–52. Aug. 2002.

[9] D. Liben-Nowell, H. Balakrishnan, et al. Analysis of the evolution of peer-to-peer systems. In *Proc. of ACM PODC Symp*. Jul. 2002.

[10] M. C. R. Mahajan and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *Proc. of IPTPS*. February 2003.

[11] S. Ratnasamy, P. Francis, et al. A scalable content-addressable network. In *Proc. of ACM SIGCOMM Conf.* ACM, Aug. 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems. In *Proc. of IFIP/ACM Middleware*. November 2001.

[13] S. Saroiu, P. K. Gummadi, et al. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking*. Jan. 2002.

[14] I. Stoica, R. Morris, et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM Conf.* ACM, Aug. 2001.

[15] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. of IPTPS*. Mar. 2002.

[16] H. Weatherspoon, T. Moscovitz, et al. Introspective failure analysis: Avoiding correlated failures in peer-to-peer systems. In *Proc. of Intl. Workshop on Reliable Peer-to-Peer Distributed Systems*. Oct. 2002.

[17] B. Y. Zhao, L. Huang, et al. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications*.