A Simple Fault Tolerant Distributed Hash Table*

Moni Naor[†‡] Udi Wieder [‡]

Abstract

We introduce a distributed hash table (DHT) with logarithmic degree and logarithmic dilation. We show two lookup algorithms. The first has a message complexity of $\log n$ and is robust under random deletion of nodes. The second has parallel time of $\log n$ and message complexity of $\log^2 n$. It is robust under spam induced by a random subset of the nodes. We then show a construction which is fault tolerant against random deletions and has an optimal degree-dilation tradeoff. The construction has improved parameters when compared to other DHT's. Its main merits are its simplicity, its flexibility and the fresh ideas introduced in its design. It is very easy to modify and to add more sophisticated protocols, such as dynamic caching and erasure correcting codes.

1 Introduction

We propose a very simple and easy to implement distributed hash table. Our construction offers logarithmic linkage, load and dilation. It can operate in a highly dynamic environment and is robust against random deletions and random spam generating nodes, in the sense that with high probability *all* nodes can locate *all* data items.

There are two commonly used methods for modelling the *occurrence* of faults. The first is the random fault model, in which every node becomes faulty with some probability and independently from other nodes. The other is the worst case model in which an adversary which knows the state of the system chooses the faulty subset of nodes. There are several models that describe the *behavior* of faulty nodes. One of them is the fail-stop model in which a faulty node is deleted from the system. Another is a spam generating model in which a faulty node may produce arbitrary false versions of the data item requested. A third model is the Byzantine model in which there are no restrictions over the behavior of faulty nodes.

In the random fault model, if we want that all nodes can access all data items, then it is necessary that the degree be at least $\log n$ and that every data item is stored by at least $\log n$ nodes[1]. Otherwise with non-negligible probability there would be nodes disconnected from the system.

1.1 Related Work

Several peer-to-peer systems are known to be robust under random deletions ([10], [8], [6]). Stoica *et al* prove that the Chord system [8] is resilient against random faults in the fail-stop model. It does not seem likely that Chord can be made spam resistant without a significant change in its design. Fiat and Saia [7] proposed a content addressable network that is robust against deletion and spam in the *worst case* scenario, when an adversary can choose which nodes fail. Clearly in this model some small fraction of the non-failed nodes would be denied from accessing some of the data items. While their solution handles a more difficult model then ours, it has several disadvantages; *(I)* it is not clear whether the system can preserve its qualities when nodes join and leave dynamically. *(II)* the message complexity is large ($\log^3 n$) and so is the linkage needed ($\log^2 n$). Most importantly their construction is very complicated. Complex constructions and algorithms increase the likelihood of errors in implementation and offer easier opportunities for an adversary to diverge from the designated protocol. In a later paper Fiat *et al* [2]

*Research supported in part by the RAND/APX grant from the EU Program IST

†Incumbent of the Judith Kleeman Professorial Chair

‡The Weizmann Institute of Science. Rehovot 76100 Israel. {naor, uwieder}@wisdom.weizmann.ac.il

[1] It is not necessary that the item be *replicated* $\log n$ times but rather that $\log n$ processors be *involved* in the storage of it.

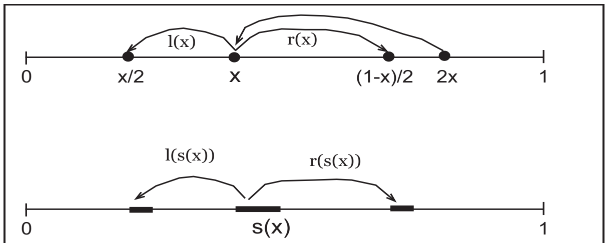

Figure 1: The upper figure shows the edges of a point in G_c. The lower shows a mapping of a segment into two smaller ones.

solve the first problem yet they do not describe a spam resistant lookup.

The construction presented here is designed using design rules which we call *continuous - discrete*. These design rules are defined and analyzed in [5], where their power is demonstrated by the suggestion of several distributed dynamic data structures. Among them is a *constant degree* DHT. The only previously known constant degree DHT is Viceroy [4] which is rather involved. The DHT described in [5] enjoys an optimal tradeoff between the degree and the dilation. A degree of $\log n$ results with a dilation of $O(\frac{\log n}{\log \log n})$ which is an improvement over previous constructions.

2 The Overlapping DHT

We describe the construction as a discretization of a continuous graph denoted by G_c. The vertex set of G_c is denoted by I and defined to be the real interval $[0, 1)$. The edge set of G_c is defined by the following functions:

$$\ell(a) \stackrel{def}{=} \frac{a}{2} \quad (1)$$

$$r(a) \stackrel{def}{=} \frac{a}{2} + \frac{1}{2} \quad (2)$$

where $a \in I$, ℓ abbreviates 'left' and r abbreviates 'right'. Note that the out-degree of each point is 2 while the in-degree is 1. Sometimes we may enhance the notation and write $r, \ell([a, b])$ meaning the image of the interval $[a, b]$ under r, ℓ.

Properties of G_c: We set some useful notations. For any two points $a, b \in I$ define $d(a, b)$ to be $|a - b|$. Let σ denote a sequence of binary digits, and σ_t denote its prefix of length t. For every point $a \in I$ define $\sigma_t(a)$ in the following manner:

$$\sigma_0(a) = a$$
$$(\sigma_t.0)(a) = \ell(\sigma_t(a))$$
$$(\sigma_t.1)(a) = r(\sigma_t(a))$$

In other words $\sigma_t(a)$ is the point reached by a walk that starts at a and proceeds according to σ_t when 0 represents ℓ and 1 represents r. The following claim justifies the name 'Distance Halving':

Claim 2.1 (distance halving property). *For all $a, b \in I$ and for all binary strings σ it holds that:*

$$d(r(a), r(b)) = d(\ell(a), \ell(b)) = \tfrac{1}{2} d(a, b) \quad (3)$$
$$d(\sigma_t(a), \sigma_t(b)) = 2^{-t} \cdot d(a, b) \quad (4)$$

For every point $a \in [0, 1)$ and t there is a string σ_t and a point a' such that $\sigma_t(a') = a$. The string σ_t could be easily calculated from a and in fact it consists of the first t bits from the binary representation of a.

Claim 2.2. *Let $a, b \in [0, 1)$ and let σ be the binary representation of a. For all t it holds that*

$$d(a, \sigma_t(b)) \leq 2^{-t}$$

Proof. Let a' be such that $\sigma_t(a') = a$; i.e. a walk that starts at a' and follows the binary representation of the prefix of length t of a, reaches a. We have $d(a, \sigma_t(b)) = d(\sigma_t(a'), \sigma_t(b))$. By Claim 2.1 it holds that $d(\sigma_t(a'), \sigma_t(b)) = 2^{-t} d(a', b) \leq 2^{-t}$. □

The Discrete graph G: We show how to construct the discrete graph G. Each node i $(1 \leq i \leq n)$ in the graph is associated with a *segment* $s(i) \stackrel{def}{=} [x_i, y_i]$. These segments should have the following properties:

Property I - The set of points $\vec{x} = x_1, x_2, \ldots, x_n$ is evenly distributed along I. Specifically we desire that every interval of length $\frac{\log n}{n}$ contains $\Theta(\log n)$ points from $\vec{x}$. The point x_i is fixed and would not change as long as i is in the network.

Property II - The point y_i is chosen such that the length of each segment is $\Theta(\frac{\log n}{n})$. It is important to notice that for $i \neq j$, $s(i)$ and $s(j)$

may *overlap*. The point y_i would be updated as nodes join and leave the system. The precise manner in which y_i is chosen and updated would be described in the next section.

The edge set of G is defined as follows. A pair of vertices i, j is an edge in G if $s(i)$ and $s(j)$ are connected in G_c or if $s(i)$ and $s(j)$ overlap. The edges of G are anti-parallel. It is convenient to think of G as an undirected graph. A point $a \in I$ is said to be *covered* by i if $a \in s(i)$. We observe the following:

1. Each point in I is covered by $\Theta(\log n)$ nodes of G. This means that each data item is stored at $\Theta(\log n)$ processors.
2. Each node in G has degree $\Theta(\log n)$.

Join and Leave: Our goal in designing the Join and Leave operations is to make sure that properties I,II remain valid. When node i wishes to join the system it does the following:

1. It chooses at random $x_i \in [0, 1)$ [2].
2. It calculates q_i which is an estimation of $\frac{\log n}{n}$.
3. It sets $y_i = x_i + q_i \mod 1$.
4. It updates all the appropriate neighbors according to the definition of the construction.
5. The neighbors may decide to update their estimation of $\frac{\log n}{n}$ and therefore change their y value.

When node i wishes to leave the system (or is detected as down) all its neighbors should update their routing tables and check whether their estimation of $\frac{\log n}{n}$ should change. If so they should change their y value accordingly. The following lemma is straight forward:

Lemma 2.3. *If n points are chosen randomly, uniformly and independently from the interval $[0, 1]$ then with probability $1 - \frac{1}{n}$ each interval of length $\Theta(\frac{\log n}{n})$ would contain $\Theta(\log n)$ points.*

If each node chooses its x-value uniformly at random from I then property-I holds. Observe that if each node's estimation of $\frac{\log n}{n}$ is accurate within a multiplicative factor then property II holds as well. The procedure for calculating q_i is very simple. Assume x_j is the predecessor of x_i along I. It is proven in [4] that with high probability[3]

$$\log n - \log\log n - 1 \leq \log\left(\frac{1}{d(x_i, x_j)}\right) \leq 3\log n$$

Conclude that node i can easily estimate $\log n$ within a multiplicative factor. Call this estimation $(\log n)_i$. A multiplicative estimation of $\log n$ implies a *polynomial* estimation of n, therefore an additional idea should be used. Let q_i be such that in the interval $[x_i, x_i + q_i]$ there are *exactly* $(logn)_i$ different x-values.

Lemma 2.4. *With high probability the number q_i estimates $\frac{logn}{n}$ within a multiplicative factor.*

The proof follows directly from lemma 2.3. Each node in the system updates its q value and holds an accurate estimation of $\frac{\log n}{n}$ at all times. Therefore property II holds at all times.

Mapping the data items to nodes: The mapping of data items to nodes is done in the same manner as other constructions of distributed hash tables (such as Chord [8], Viceroy [4] and CAN [6]). First data items are mapped into the interval I using a hash function. Node i should hold all data items mapped to points in $s(i)$. The use of consistent hashing [3] is suggested in Chord [8]. Note that all nodes holding the same data item are connected to one another so they form a clique. If a node storing a data item is located, then other nodes storing the same data item are quickly located as well. This means that accessing different copies of the same data item in parallel can be simple and efficient. It suggests storing the data using an erasure correcting code, (for instance the digital fountains suggested by Byers *et al* [1]) and thus avoid the need for replication. The data stored by any small subset of the nodes would suffice to reconstruct the date item. Weatherspoon and Kubiatowicz [9] suggest that an erasure correcting code may improve significantly the bandwidth and storage used by the system.

[2] It may be that x_i is chosen by hashing some i.d. of i. In this case it is important that the hash function distribute the x values evenly.

[3] The term 'with high probability' (w.h.p) means with probability $1 - n^{-\epsilon}$

3 The Lookup Operation

The lookup procedure emulates a walk in the continuous graph G_c. Assume that processor i wishes to locate data item V and let $v = h(V)$ where h is a hash function, i.e. data item V is stored by every processor which covers the point v. Let $z_i = \frac{x_i + y_i}{2}$ and let σ be the binary representation of z_i. Claim 2.2 states that $\sigma_t(v)$ is within the interval $[z_i - 2^{-t}, z_i + 2^{-t}]$. Conclude that when $t = \log n - \log\log n + O(1)$ it holds that $\sigma_t(v) \in s(i)$. Let t to be the minimum integer for which $\sigma_t(v) \in s(i)$. Call the path between $\sigma_t(v)$ and v the *canonical path* between v and $s(i)$. This gives rise to a natural lookup algorithm. The canonical path exists is G_c, yet by the definition of G, if (a, b) is an edge in G_c, a is covered by i and b is covered by j then the edge (i, j) exists in G. This means that the canonical path can be *emulated* by G.

Simple Lookup: Every point in I is covered by $\Theta(\log n)$ nodes. This means that when node i wishes to pass a message to a node covering point $z \in I$ it has $\Theta(\log n)$ *different* neighbors that cover z. In the Simple Lookup it chooses *one* of these nodes at random and sends the message to it.

Theorem 3.1. *Simple Lookup has the following properties:*

1. *The length of each lookup path is at most* $\log n + O(1)$. *The message complexity is* $\log n + O(1)$.

2. *If i is chosen at random from the set of nodes and v is chosen at random from I, then the probability a given processor participates in the lookup is* $\Theta\left(\frac{\log n}{n}\right)$.

Proof Sketch: The proof of statement (1) is immediate. To show the correctness of statement (2) we prove the following: Fix a processor i. The probability processor i participates in the k^{th} step of the routing $1 \le k \le \log n$ is $\Theta(\frac{1}{n})$. Summing up over k yields the result. This statement is proved by induction on k. □

Theorem 3.2. *If each node is faulty independently with fixed probability p, then for sufficiently low p (which depends entirely on the parameters chosen when constructing G), with high probability each surviving node can locate every data-item.*

Proof. We prove the following claim:

Claim 3.3. *If p is small enough, then w.h.p every point in I is covered by at least one node.*

Proof. Assume for simplicity that $x_1 < x_2 < \cdots < x_n$. Each point in an interval $[x_i, x_{i+1}]$ is covered by the *same* set of $\Theta(\log n)$ nodes. Call this set S_i. We have

$$\Pr[\text{ All nodes in } S_i \text{ were deleted }] = p^{\Theta(\log n)}$$

Therefore for sufficiently small p this probability is smaller than n^{-2}. Applying the union bound over all i yields that with probability greater than $1 - \frac{1}{n}$ every point in I is covered by at least one node. It is important to notice that for an arbitrary value of p it is possible to adjust the q values, so that each point in I is covered by sufficiently many nodes, and the claim follows. □

For every edge (a, b) in G_c there exists at least one edge in G whose nodes cover a and b, therefore the canonical path could be emulated in G and the simple lookup succeeds. We stress that after the deletions the lookup still takes $\log n$ time and $\log n$ messages. Furthermore the average load induced on each node does not increase significantly. □

Spam Resistant Lookup: Assume that a faulty node may generate arbitrarily false data items. We wish to show that every node can find all *correct* data items w.h.p. Just as in the simple lookup, the spam resistant lookup between i and v emulates the canonical path between $s(i)$ and v. The main difference is that now when node i wishes to pass a message to a node covering point a it will pass the message to *all* $\Theta(\log n)$ nodes covering a. At each time step each node receives $\Theta(\log n)$ messages, one from each node covering the previous point of the path. The node sends on a message only if it were sent to it by a *majority* of nodes in the previous step.

Theorem 3.4. *The spam resistant lookup has the following properties:*

1. *With high probability all surviving nodes can obtain all correct data items.*
2. *The lookup takes (parallel) time of* $\log n$.
3. *The lookup requires* $O(\log^2 n)$ *messages in total.*

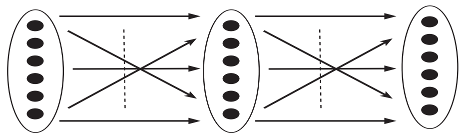

Figure 2: The message is sent through *all* the nodes covering the canonical path.

Proof. Statements $(2, 3)$ follow directly from the definitions of the spam resistant lookup. Statement (1) follows from the following:

Claim 3.5. *If each node fails with probability p, then for sufficiently small p (which depends entirely on the parameters chosen when constructing G) it holds that with high probability every point in I is covered by a majority of non-failed processors.*

The proof of claim 3.5 is similar to that of claim 3.3. Now the proof of theorem 3.4 is straight forward. It follows by induction on the length of the length of the path. Every point of the canonical is covered by a majority of good nodes, therefore every node along the path would receive a majority of the authentic message. It follows that with high probability *all* nodes can find *all* true data items. □

The easy proofs of theorems 3.2 and 3.4 demonstrate the advantage of designing the algorithms in G_c and then migrating them to G. Proving the robustness of G_c is a straight forward argument.

4 Reducing the Dilation

We now show how to decrease the lookup length and the congestion by increasing the degree. For any integer $c \geq 2$ construct a continuous graph G_c with edges defined by the following functions:

$$f_i(y) = \frac{y}{c} + \frac{i}{c} \; (i = 0, 1, \ldots, c-1)$$

The equivalent of Claim 2.1 is

$$d(f_i(y), f_i(z)) = \frac{1}{c} d(y, z)$$

and the equivalent of Claim 2.2 is $d(y, \sigma_t(z)) \leq c^{-t}$. Therefore:

Theorem 4.1. *A discretization of G_c would result with a graph of degree $\Theta(c \log n)$ and with dilation $\log_c n$.*

Two interesting options are setting $c = \log n$ or $c = n^\epsilon$ (for some constant ϵ), as the first results with a lookup length of $\frac{\log n}{\log \log n}$, and the second with a lookup length of $O(1)$. It is worth noting that the same analysis of Theorem 3.1 shows that for each choice of c, the probability a processor participates in a lookup is $\Theta(\frac{\log_c n}{n})$.

Optimizing the Degree-Dilation Tradeoff: We show how to achieve a dilation of $\Theta(\frac{\log n}{\log \log n})$ while maintaining a degree of $\Theta(\log n)$, thus improving the lookup and congestion while maintaining the same degrees as Chord. First set $c = \log n$. The previous construction yields a graph with the desired dilation yet with a degree of $\Theta(\log^2 n)$. The reduction of the degree is achieved by connecting each processor to only *one* other processor for each projection of its segment. More formally, for each $0 \leq j \leq \log n$ the length of $f_j([x_i, y_i])$ is $\Theta(\frac{1}{n})$, therefore it is *covered* by $\Theta(\log n)$ different processors. processor i is connected to *exactly one* randomly chosen processor whose segment covers $f_j([x_i, y_i])$. As before there exists a link between processor i and all processors with segments that overlap $s(i)$. Lookup is done in the same manner as Simple Lookup; i.e. the route a message takes emulates the canonical path of the continuous graph. Clearly the maximum degree of the construction is $\Theta(c + \log n) = \Theta(\log n)$ and when there are no faults the dilation is only $\Theta(\frac{\log n}{\log \log n})$. It is left to show that this construction remains fault tolerant under random deletions of processors and connections.

As before we assume that each processor fails with some fixed probability p. Assume that processor i tries to move the message from point $a \in s(i)$ to point $f(a)$. Processor i should do the following

1. If the processor which is connected to i and covers $f(a)$ is alive then i moves the message to it.
2. If i fails moving the message to $f(a)$ then it picks at random a processor covering a and moves the message to it.

We need to bound the number of times Step (2) occurs, i.e. the number of hops in which the message remains in the same spot on the continuous graph.

Lemma 4.2. *W.h.p the number of hops a message stays in point a in G_c, (before moving to $f(a)$), is dominated by a geometric random variable with a constant success probability.*

Proof Sketch: Assume processor i covers a. There are $\Theta(\log n)$ processors which covers $f(s(i))$ (and therefore cover $f(a)$) out of which i chose at random one. W.h.p out of the $\Theta(\log n)$ processors which cover $f(a)$ a constant fraction survived. Therefore the probability a processor that covers a is connected to a live processor that covers $f(a)$ is at least a constant and that is independent from the choices of other processors. Conclude that the number of hops a message remains in a is dominated by a geometric random variable $\square$

Theorem 4.3. *The expected dilation of a message after faults is $\Theta(\frac{\log n}{\log\log n})$. The actual dilation is at most $\Theta(\frac{\log n}{\log\log n})$ with probability $1 - n^{-\frac{1}{\epsilon \log\log n}}$.*

Proof Sketch: The expectation of a geometric random variable is constant. Lemma 4.2 and the linearity of expectation implies that even after random faults the expected dilation of a message is $\Theta(\frac{\log n}{\log\log n})$.

The total number of hops a message remains in the same spot is at most the sum of $\Theta(\frac{\log n}{\log\log n})$ independent geometric variables. Standard use of tail bounds for hypergeometric distributions yields the second assertion. $\square$

5 Extensions

Dynamic Caching: The simplicity of the construction implies that it is easy to modify and add protocols. In [5] we show a simple protocol that performs dynamic caching of a popular data items, thus relieving hot spots in the system. The protocol can provably prevent the existence of hot spots in the network. The protocol was designed for a constant degree non overlapping DHT. It is rather straightforward to modify it for the overlapping DHT.

Expander Graphs: It is shown in [5] that similar techniques could be used to build a graph that is guaranteed to be and expander. The idea is to use the Gabber Galil continuous expander over $[0, 1) \times [0, 1)$ and then compose it into cells using a Voronoi diagram.

6 Future Work

The main challenge ahead is to prove robustness against a worst case scenario, where an adversary chooses which nodes fail. We believe that a slight variation of the construction might be able to route messages successfully in the worst case model as well.

None of the known constructions (including [2],[7]) can handle the case in which an adversary controls the nodes *prior to their insertion*. This means that an adversary may control the actual construction of the network, and thus cause faults that otherwise would have been beyond its capability. It seems likely that robustness against such an adversary would require the use of cryptographic means.

References

[1] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, pages 56–67, 1998.

[2] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Symposium on Discrete Algorithms (SODA)*, 2002.

[3] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[4] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *PODC*, 2002.

[5] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. *www.wisdom.weizmann.ac.il/ ˜uwieder*, 2002.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc ACM SIGCOMM*, pages 161–172, San Diego CA, Augost 2001.

[7] J. Saia, A. Fiat, S. Gribble, A. R. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *First International Workshop on Peer-to-Peer Systems*, MIT Faculty Club, Cambridge, MA, USA, 2002.

[8] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[9] H. Weatherspoon and J. D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *First International Workshop on Peer-to-Peer Systems*, MIT Faculty Club, Cambridge, MA, USA, 2002.

[10] B.Y. Zhao and J. Kubiatowicz. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB CSD 01-1141, University of California at Berkeley, Computer Science Department, 2001.