VHash: A Voronoi-Based Multidimensional Distributed Hash Table

Brendan Benshoof Andrew Rosen
Department of Computer Science, Georgia State University
bbenshoof@cs.gsu.edu rosen@cs.gsu.edu

***Abstract*—Distributed Hash Tables are used as a tool to generate overlay networks for P2P networks. Current DHT techniques are not designed to take the nature of the underlying network into account when organizing the overlay network. Current DHT networks assign nodes locations in a ring or tree, limiting the ability of these networks to be more efficient. A DHT technique that allows for efficient construction of an overlay network that takes into account the real underlying network would allow for higher performance and faster P2P networks. We present VHash as a spacial DHT based on approximate Delaunay Triangulation to integrate distance information between nodes into overlay network topology. VHash allows for the creation of P2P networks with faster record lookup time, storage, and maintenance with a geographically diverse set of nodes.**

I. Introduction

A Distributed Hash Table is used provide an overlay network for many P2P applications. State of the art DHT techniques are built on tree or log-ring structures to ensure that the routing distance is $O(lg(n))$ hops between nodes. These topologies, while sufficient in reasonably local networks, do not take into account the lengths or latencies of routes defined by the topology, assuming that every hop has similar latency and throughput. For a global network, a more intelligent means of generating a dynamic overlay network with efficient routing, storage, and backups is needed for future P2P applications.

We present VHash as a DHT designed to take inter-node latency information into account when generating an overlay on a massive scale. VHash creates a

Our paper presents the following:

- We define how VHash functions.
- We show how file sharing and fault tolerance works.
- We do some simulations to prove it works.
- We present the related work.
- We present our future work.

Two difficulties that arise in attempting to use Voronoi Tessellation for constructing a DHT.

II. VHash

VHash is generalization of ring-based DHTs like Chord [1], Pastry , and Symphony from 1-dimensional keyspaces to keyspaces with any number of dimensions.

VHash was created to allow for spacial representations to be mapped to hash locations, a feature lacking in many current distributed hash tables. In particular, we aimed to construct a mechanism for creating a more efficient global scale DHT built on a minimal latency overlay. Rather than focus on minimizing the amount of hops required to travel from point to point we wish to minimize the time required for a message to reach its recipient. VHash actually has a worse worst case hop distance ($O(\sqrt[d]{n})$) than other comparable distributed hash tables ($O(lg(n))$). However, VHash can route messages as quickly as possible rather than traveling over a grand tour that an overlay network may describe in the real world.

The naive method of doing so is to assign coordinates to servers based on the geographic location of nodes. More complex approaches would approximate a minimum latency space based on inter-node latency. VHash can be considered a generalized

extension of VoroNet [2]. Algorithm 1 describes the process for performing a minimum latency embedding using VHash.

Chord, if it were generalized out to d dimensions without finger tables would have a lookup time of $\sqrt[d]{n}$.

- Assignment of keys and responsibility. IN Chord it is the successor, Pastry is closest to id.
- Pastry's locality heuristic assumes Euclidean space, namely triangle equality holds. How do we get around that. Pastry also assumes magic distance function is provided to determine distance between two nodes, however distance is not embedded in the keyspace (which is randomly distributed) but rather the "proximity space" built off of bootstrapped assumetions (node we are using to joining the ring is close to the joining node in the proximity space, and the proximity assumptions for a network were good prior to the join operation. The key idea is that at each hop messages travel closer to the destination, while minimizing the metric distance traveled (such as num hops). Expected distance traveled grows exponentially with each hop. their results showed that the metric distance traveled by routed message over pastry was 30%-40% greater than the distance between the source and destination. Has to uyse second stage when joining to maintain locality. Total join cost in messages is $\approx 3 \cdot 2^b \cdot \log_{2^b} N$
- Cost of computing the Voronoi region. We get around this by accurately approximating it and letting our fault tolerance mechanisms handle any discrepancies until the network repairs itself.

A. Key Generation - Mapping a Node to Coordinates

B. Memory Overhead

In order to route, a node maintains a routing table consisting of the nodes it borders a region with. Unlike other DHTs, which keep routing tables of a set size, VHash's routing table depends on the nodes that it shares a border with, which is on average $3^d - 1$

Algorithm 1 VHash Minimum Latency Embedding

1: d is the dimensions of the hash space
2: seed the space with $d + 1$ nodes at random locations
3: A node n wishes to join the network
4: n pings a random subset of peers to find latencies L
5: Normalize L onto (0.0,1.0) to yield L_N
6: Choose position p that minimizes

$$\sum_{i \in peers} (L_N[i] - dist(p, i))^2$$

7: Re-evaluate location periodically

Consider a d-dimensional toroidal space. Let some arbitrary point A be the center of this space[1]. A hypercube that surrounds the space would be defined by $3^d - 1$ points citation [2]

C. Toroidal Distance Equation

Given two vector locations $\vec{a}$ and $\vec{b}$ on a d dimensional unit toroidal hypercube:

$$distance = \sqrt[|d|]{\sum_{i \in d} (\min(|\vec{a}_i - \vec{b}_i|, 1.0 - |\vec{a}_i - \vec{b}_i|))^2}$$

D. Mechanism

VHash maps nodes to a d dimension toroidal unit space overlay. This is essentially a hypercube with wrapping edges. The toroidal property makes visualization difficult but allows for a space without a sparse edge, as all nodes can translate the space such that they are at the center of the space. In effect, each node views itself at the center of the graph.

VHash nodes are responsible for the address space defined by their Voronoi region. This region is defined by a list of peer nodes maintained by the node. A minimum list of peers is maintained such that the node's Voronoi region is well defined. The

[1] Any point in a toroidal space can view itself as it's center
[2] Is this rigorous enough?

Fig. 1: The starting network topology. The blue lines demark the Voronoi edges, while the red lines connecting the nodes correspond to the Delaunay Triangulation edges and one-hop connections.

links connecting the node to its peers correspond to the links of a Delaunay Triangulation. One such possible network is shown on Figure 1.

E. Relation to Voronoi Diagrams and Delaunay Triangulation

VHash does not strictly solve Voronoi diagrams [3], as the toroidal nature of the space preclude the traditional means of solving for Voronoi regions. However, VHash's peer management approximates a topology with similar properties. An online algorithm (Algorithm 2 maintains the set of peers defining the node's Voronoi region. The set of peers required to define a node's Voronoi Region corresponds to a solution to the dual Delaunay Triangulation.

Algorithm 2 VHash Greedy Peer Selection

$Candiates$ is the set of candidate peers
$Peers$ is the set of this node's peers
$Canidates$ is sorted by each node's closeness to this node
The closest member of $Canidates$ is popped and added to $Peers$
for all n in $Canidates$ **do**
    c is the midpoint between this node and n
    if Any node in $Peers$ is closer to c than this node **then**
        reject n as a peer
    else
        Add n to $Peers$
    end if
end for

F. Messages

Maintenance and joining are handled by a simple periodic mechanism. A notification message consisting of a node's information and active peers is the only maintenance message. All messages have a destination hash location which is used to route them to the proper server. This destination can be the hash location of a particular node or the location of a desired record or service. The message is received by the node responsible for the location. Services running on the DHT define their own message contents, such as commands to store and retrieve data.

G. Message Routing

Messages are routed over the overlay network using a simple algorithm (Algorithm 3). When routing a message to an arbitrary location, a node calculates who's Voronoi region the message's destination is in amongst the itself and its peers. If the destination falls within its own region, then it is responsible and handles the message accordingly. Otherwise, the node forwards the message to the closest peer to the destination location. This process describes[3] a pre-computed and cached A* routing algorithm [4] .

Algorithm 3 Vhash Routing

P_0 is this node's set of peers
N is this node
m is a message addressed for L
$Forwards$ is the set $P_0 \cup N$
find C: member of $Forwards$ which has the shortest distance to L
if C is N **then**
    N is the responsible party.
    Handle m
else
    Forward m to C for handling or further routing
end if

H. Joining and Maintenance

Joining the network is a straightforward process. A new node first learns the location of at least one member of the network to join. The joining node

[3]Wording. Is equivalent a better word?

then choses a location in the hash space either at random or based on a problem formulation (for example, based on geographic location or latency information).

After choosing a location, the joining node sends a "join" message to its own location via the known node. The message is forwarded to the current owner of that location who can be considered the "parent" node. The parent node immediately replies with a maintenance message containing its full peer list. This message is sent to the joining node, who then uses this to begin defining the space it is responsible for.

The joining node's initial peers are a subset of the parent and the parent's peers. The parent adds the new node to its own peer list and removes all his peers occluded by the new node. Then regular maintenance propagates the new node's information and repairs the overlay topology. This process is described by Algorithm 4.

I. Eclipse and Sybil Attacks

Algorithm 4 Vhash Join

1: new node N wishes to join and has location L
2: N knows node x to be a member of the network
3: N sends a request to join, addressed to L via x
4: node $Parent$ is responsible for location L and receives the join message
5: $Parent$ sends to N its own location and list of peers
6: $Parent$ integrates N into its peer set
7: N builds its peer list from N and its peers
8: regular maintenance updates other peers

Each node in the network performs maintenance periodically by a maintenance message to its peers. The maintenance message consists of the node's information and the information on that node's peer list. When a maintenance message is received, the receiving node considers the listed nodes as candidates for its own peer list and removes any occluded nodes (Algorithm 2).

When messages sent to a peer fail, it is assumed the peer has left the network. The leaving peer is removed from the peer list and candidates from the set of 2-hop peers provided by other peers move in to replace it. Maintenance is described by Algorithms 5 and 6. Figures **??**, 3, and 4 illustrate the joining processing.

Algorithm 5 VHash Maintenance Cycle

1: P_0 is this node's set of peers
2: T is the maintenance period
3: **while** Node is running **do**
4: **for all** node n in P_0 **do**
5: Send a Maintenance Message containing P_0 to n
6: **end for**
7: Wait T seconds
8: **end while**

Algorithm 6 VHash Handle Maintenance Message

1: P_0 is this node's set of peers
2: Receive a Maintenance Message from peer n containing its set of peers:P_n
3: **for all** Peers p in P_n **do**
4: Consider p as a member of P_0
5: **if** p should join P_0 **then**
6: Add p to P_0
7: **for all** Other peers i in p **do**
8: **if** i is occluded by p **then**
9: remove i from P_0
10: **end if**
11: **end for**
12: **end if**
13: **end for**

There is no function for a "polite" exit from the network. VHash assumes nodes will fail and the difference between an intended failure and unintended failure is unnecessary. The only issue this causes is that node software should be designed to fail totally when issues arise rather then attempt to fulfill only part of its responsibilities.

J. Data Storage and Backups

The primary goal of a DHT is to provide a distributed storage medium. We extend this idea

Fig. 2: Here, a new node is joining the networks and has established that his position falls in the the yellow shaded Voronoi region.

Fig. 3: The network topology after the new node has finished joining.

to distribute work and information among nodes using the same paradigm. Resources in the network, be it raw data or assigned tasks, are assigned hash locations. The node responsible for a given hash location is responsible for the maintenance of that resource. When a node fails, its peers take responsibility of its space. Thus it is important to provide peers with frequent backups of a node's assigned resources. That way, when a node fails, its peers can immediately assume its responsibilities.

When a resource is to be stored on the network, it is assigned a hash location. The hash locations assigned could be random, a hash of an identifier, or have specific meaning for an embedding problem. The node responsible for that resource's hash location stores the resource.

A resource is accessed by contacting the node responsible for the resource. However, the requester generally has no idea which node is responsible for any particular resource. The data request messsage is addressed to the location corresponding to the resource, rather than the node responsible for that location. The message is forwarded over the overlay network, each hop bringing the node closer until it reaches the responsible node, who sends the resource or an error if the resource does not exist.

Some options are immediately apparent for dealing with wasted storage space. A system that is primarily read driven can record the time of the last read or a frequency of reads such that resources that are not read often enough are deleted after a certain period of time. If a system is write driven, allow the resource to be assigned a time to live, which can be updated as needed.

A node periodically sends a message containing backups of the resources for which it became newly responsible for to each of its peers. To minimize bandwidth and time wasted by backups, the node should only send the records changed since last backup.

Fig. 4: The topology immediately after the new node leaves the network. After maintenance takes place, the topology repairs itself back to the configuration shown in Figure 1.

III. Related Work

Voronet stuff here.

IV. Conclusions and Future Work

Apply this to MANET.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM Computer Communication Review*, vol. 31, pp. 149–160, ACM, 2001.

[2] O. Beaumont, A.-M. Kermarrec, L. Marchal, and E. Rivière, "Voronet: A scalable object network based on voronoi tessellations," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–10, IEEE, 2007.

[3] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, pp. 345–405, Sept. 1991.

[4] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.