

2015-07-10

DHT Distributed Computing

└ Introduction

└ Distributed Computing and Challenges

└ Challenges of Distributed Computing

Challenges of Distributed Computing

Distributed Computing platforms should be:

- Scalable: The larger the network, the more resources need to be spent on maintaining and organizing the network.

Remember, computers aren't telepathic. There's always an overhead cost. It will grow. The challenge of scalability is designing a protocol that grows this organizational cost at an extremely slow rate. For example, a single node keeping track of all members of the system might be a tenable situation up to a certain point, but eventually, the cost becomes too high for a single node.

2015-07-10

DHT Distributed Computing

└ Introduction

└ Distributed Computing and Challenges

└ Challenges of Distributed Computing

Challenges of Distributed Computing

Distributed Computing platforms should be:

- Scalable: The larger the network, the more resources need to be spent on maintaining and organizing the network.
- Fault Tolerant: As you add more machines, you need to be able to handle the increased risk of hardware failure.

Hardware failure is a thing that can happen. Individually the chances are low, but this becomes high when we're talking about millions of machines. Also, what happens in a P2P environment.

2015-07-10

DHT Distributed Computing

└ Introduction

└ Distributed Computing and Challenges

└ Challenges of Distributed Computing

Challenges of Distributed Computing

Distributed Computing platforms should be:

- Scalable: The larger the network, the more resources need to be spent on maintaining and organizing the network.
- Fault Tolerant: As you add more machines, you need to be able to handle the increased risk of hardware failure.
- Load Balanced: Tasks need to be evenly distributed among all the workers.

If we are splitting the task into multiple parts, we need some mechanism to ensure that each worker gets an even (or close enough) amount of work.

2015-07-10

DHT Distributed Computing

└ Introduction

└ What Are Distributed Hash Tables?

└ How Does It Work?

How Does It Work?

We're regular in general (most time), but binary:

- DHT's organize a set of nodes, each identified by an ID (either IP or key).
- Nodes are responsible for the keys that are closest to their IDs.
 - Nodes maintain a list of other peers in the network.
 - Usually a few "high" value nodes in the network.
- Nodes that store a very simple routing algorithm to find a node responsible for any given key.

We use ID for nodes and keys for data so we always know our context.

2015-07-10
DHT Distributed Computing
└─ Introduction
└─ What Are Distributed Hash Tables?
└─ Strengths of DHTs

Strengths of DHTs

DHTs are designed for large P2P applications, which means they need to be just *right*.

- Scalability: Each node knows a small subset of the entire network.
- Self-Heal: Each node operations impact very few nodes.
- Fault-Tolerance: The network is decentralized.
- Load-Balancing: DHTs are designed to handle *hot* data.

A consistent hashing ensures that nodes and data are close to evenly distributed.

Nodes are responsible for the data closest to it.

- Remember to mention Napster.
- Distributed Hash Tables were designed to be used for completely decentralized P2P applications involving millions of nodes.
- As a result of the P2P focus, DHTs have the following qualities.
- Scalability
 - The subset each node knows is such that we have expected $\lg(n)$ lookup
- Fault-Tolerance
 - Because Joins and node failures affect only nodes in the immediate vicinity, very few nodes are impacted by an individual operation.
- Load Balancing
 - The space is large enough to avoid Hash collisions

2015-07-10
DHT Distributed Computing
└─ Introduction
└─ Why DHTs and Distributed Computing
└─ Uses For DHT Distributed Computing

Uses For DHT Distributed Computing

- Enabling Parallel Computations
- Real-time computations
- Content aggregation
- Network state (e.g., network health)
- Key-value stores (e.g., key-value stores)
- Key-value stores (e.g., key-value stores)
- Can be used in other P2P context as a more traditional deployment.

- Need notes here
- Define Monte-Carlo Markov Chain

2015-07-10
DHT Distributed Computing
└─ Background
└─ The Components and Terminology
└─ Attributes of DHT

Attributes of DHT

- A distance and midpoint function.
- A distance definition.
- A peer management strategy.

- There needs to be a way to establish how far things are from one another. Once we have a distance metric, we define what we mean when we say a node is responsible for all data *close* to it.
- We'll go into more detail about the difference between Distance and Closeness in Chord

2015-07-10
DHT Distributed Computing
└─ Background
└─ The Components and Terminology
└─ Terms and Variables

Terms and Variables

- Network size is a node.
- Node and ID are generated on the host, usually SHA1.
- Peers are made up of:
 - Short Peers: The neighboring nodes that define the network's topology.
 - Long Peers: Distant nodes.
- We'll call the node responsible for a key the root of the key.

- SHA1 is being depreciated.
- Short peers are actively maintained, long peers replaced gradually and are not actively pinged.
- We use root as it's a topology agnostic term.

2015-07-10

DHT Distributed Computing

- Background
 - The Components and Terminology
 - Functions

Functions

findval(key) Finds the node responsible for a given key

put(key, value) Stores value at the node responsible for key

where key = hash(key)

get(key) Returns the value associated with key

- There is usually a delete function as well, but it's not important.
- All nodes use the same general lookup: Forward the message to the node closest to *key*

2015-07-10

DHT Distributed Computing

- Background
 - Example DHT: Chord
 - Chord

Chord

- Ring Topology
 - Store Finger, predecessor and successor in the ring
 - Responsible for keys between their predecessor and their own
 - Long Finger: long i nodes, when the node at index i is the previed is

$query \rightarrow 2^{i-1}$ mod m , $1 \leq i \leq m$

- Chord is a favorite because we can draw it.
- Draw a Chord network on the wall?
- node r is our root node.
- i is the index on the list
- English for the equation, the long peers double in distance from the root node, allowing us to cover at least half the distance to our target in a step
- In this way, we can achieve an expected $\lg n$ hops.

2015-07-10

DHT Distributed Computing

- Completed Work
 - VHash
 - Goals

Goals

Visualizing space from two related ideas:

- We wanted a way for distributed latency by embedding in into the existing topology
- A lot more other aspects are needed.

Most DHTs optimize routing for the number of hops, rather than latency.

2015-07-10

DHT Distributed Computing

- Completed Work
 - VHash
 - Goals

Goals

Visualizing space from two related ideas:

- We wanted a way for distributed latency by embedding in into the existing topology
- We wanted to create a DHT based off of Voronoi triangulation. Unfortunately

We discovered a mapping between Distributed Hash Tables and Voronoi/Delaunay Triangulations.

2015-07-10
DHT Distributed Computing
Completed Work
VHash
Goals

Goals

Which spring from two related ideas:

- The natural way for distributed systems to embed in the existing world.
- The natural way to DHT based off of Voronoi tessellation.
- Unfortunately
- Existing algorithms for this problem don't really work.

I lie, they do exist, but they all are “run the global algorithm on your local subset. And if we move out of or above 2D Euclidean space, as Brendan wanted to, no fast algorithms exist at all. We quickly determined that solving was never really a feasible option. So that leaves approximation. A distributed algorithm would be helpful for some WSNs solving the boundary coverage problem.

2015-07-10
DHT Distributed Computing
Completed Work
VHash
Goals

Goals

Which spring from two related ideas:

- The natural way for distributed systems to embed in the existing world.
- The natural way to DHT based off of Voronoi tessellation.
- Unfortunately
- Existing algorithms for this problem don't really work.
- Existing approximation algorithms were inadequate.

Simple approximations have no guarantees of connectivity, which is very bad for a routing topology. Better algorithms that existed for this problem technically ran in constant time, but had a prohibitively high sampling. So to understand what I'm talking about here, let's briefly define what a Voronoi tessellation is.

2015-07-10
DHT Distributed Computing
Completed Work
VHash
Voronoi Tessellation

Voronoi Tessellation

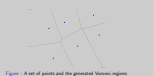


Figure: A set of points and the generated Voronoi regions

Define

- A Voronoi tessellation or Voronoi diagram divides a space into regions, where each region encompasses all the points closest to Voronoi generators (point).
- Voronoi generators
- Voronoi Region
- Voronoi Tessellation/ Diagram

2015-07-10
DHT Distributed Computing
Completed Work
VHash
Delaunay Triangulation

Delaunay Triangulation




Figure: The same set of nodes with their corresponding Delaunay Triangulation

Define

- Delaunay Triangulation is a triangulation of a set of points with the following rule:
- No point falls within any of the circumcircles for every triangle in the triangulation,
- The Voronoi tessellation and Delaunay Triangulation are dual problems
 - Solving one yields the other.
 - We can get the Voronoi diagram by connecting all the centers of circumcircles.
 -

2015-07-10

DHT Distributed Computing

Completed Work

VHash

VHash

VHash

• Hierarchical Distributed Hash Table based on this relationship.

• Uses low approximation to value for Delaunay neighbors, called DVH.

• Finding neighbors across the gossip-based protocol.

• Routing speed is $O(\sqrt{n})$.

• Memory Cost

• Worst case: $O(n)$

• Expected maximum size: $O(\sqrt{n})$

- d is number of dimensions, but we optimize over latency so that's deceptive
- Gossip: every cycle, a node chooses a peer and swaps peerlist information, then reruns the approximation.
- This would only happen in a contrived case and would give $O(1)$ routing
- We found a paper that proved $\Theta(\frac{\log n}{\log \log n})$ is the expected maximum degree of a vertex in a Delaunay Triangulation.

2015-07-10

DHT Distributed Computing

Completed Work

VHash

DGVH Heuristic

DGVH Heuristic

1. Given node n and its list of candidates.

2. peers = empty set that will contain its one-hop peers.

3. Sort candidates in ascending order by each node's distance to n .

4. Remove the first member of candidates and add it to peers.

5. For all c in candidates do

6. mid = the midpoint between n and c .

7. If this node is peer closer to n than n then in peers.

8. Reject c as a peer

9. else

10. Remove c from candidates

11. Add c to peers

12. end if

13. end for

- The majority of Delaunay links cross the corresponding Voronoi edges.
- We can test if the midpoint between two potentially connecting nodes is on the edge of the voronoi region.
- This intuition fails if the midpoint between two nodes does not fall on their voronoi edge.

2015-07-10

DHT Distributed Computing

Completed Work

VHash

DGVH Heuristic

DGVH Heuristic

1. Given node n and its list of candidates.

2. peers = empty set that will contain its one-hop peers.

3. Sort candidates in ascending order by each node's distance to n .

4. Remove the first member of candidates and add it to peers.

5. For all c in candidates do

6. mid = the midpoint between n and c .

7. If this node is peer closer to n than n then in peers.

8. Reject c as a peer

9. else

10. Remove c from candidates

11. Add c to peers

12. end if

13. end for

1. 'n' is the "myself" node, and the location we are seeking to find the peers of.
2. peers is a set that will build the peerlist in
3. We sort the candidates from closest to farthest.
4. The closest candidate is always guaranteed to be a peer.
5. Iterate through the sorted list of candidates and either add them to the peers set or discard them.
6. We calculate the midpoint between the candidate and the center 'n'.
7. If this midpoint is closer to a peer than 'n', then it does not fall on the interface between the location's Voronoi regions.
8. in this case discard it
9. otherwise add it the current peerlist

2015-07-10

DHT Distributed Computing

Completed Work

VHash

DGVH Heuristic

DGVH Heuristic

1. Given node n and its list of candidates.

2. peers = empty set that will contain its one-hop peers.

3. Sort candidates in ascending order by each node's distance to n .

4. Remove the first member of candidates and add it to peers.

5. For all c in candidates do

6. mid = the midpoint between n and c .

7. If this node is peer closer to n than n then in peers.

8. Reject c as a peer

9. else

10. Remove c from candidates

11. Add c to peers

12. end if

13. end for

Theoretically, this is worst case $O(n^2)$.
However in practice, this is $O(n \log(n)(\text{sorting}) + kn)$ where k is the number of Delaunay peers.
We are well aware that 2d-euclidean algorithms exist in $O(n \log(n))$ time. While we use that use case to communicate the algorithm, it is intended to be used in more exotic spaces.
realistically k is the function of the metric space and is $O(1)$ for euclidean spaces.

2015-07-10

DHT Distributed Computing

- Completed Work
 - VHash
 - Conclusions

Conclusions

- DGVH is a simple approximation for Delaunay Triangulation that guarantees a fully connected graph.
- VHash can optimize over a metric such as binary and achieve superior routing speeds as a result.

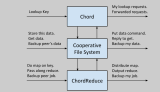
- DGVH is of similar complexity to picking k -nearest node or nodes in distance k .
- other methods don't guarantee full connectivity
- It caps out at $O(n^2)$ complexity, no matter how many dimensions or complexities of the metric space (unless calculating distance or midpoint is worse than $O(1)$)
- for example This means you can use in it an 100-dimensional euclidean space in $O(n^2)$ time rather than $O(n^{50})$ time (maybe we should have opened with this...)

2015-07-10

DHT Distributed Computing

- Completed Work
 - ChordReduce
 - System Architecture

System Architecture




- Chord [7], which handles routing and lookup.
- The Cooperative File System (CFS) [3], which handles storage and data replication.
- The MapReduce layer.
- Files are split up, each block given a key based on their contents.
- Each block is stored according to their key.
- The hashing process guarantees that the keys are distributed near evenly among nodes.
- A keyfile is created and stored where the whole file would have been found.
- To retrieve a file, the node gets the keyfile and sends a request for each block listed in the keyfile

2015-07-10

DHT Distributed Computing

- Completed Work
 - ChordReduce
 - Mapping Data

Mapping Data




- this process is recursive

2015-07-10

DHT Distributed Computing

- Completed Work
 - ChordReduce
 - Reducing Results of Data

Reducing Results of Data



The paths here are arbitrary edges that I came up with for the example.

2015-07-10
DHT Distributed Computing
Completed Work
ChordReduce
Experiment Details

Experiment Details

Our test was a Monte Carlo approximation of π .

- Map jobs were sent to randomly generated hash domains.
- The ratio of hits to generated points approximates $\pi/4$.
- Reducing the ratio with a ratio of combining the test data.

Figure: The ratio of hits to generated points $\pi/4$ and a histogram of the ratio of hits to generated points.

- Experiment had these goals
 - ChordReduce provided significant speedup during a distributed job.
 - ChordReduce scaled.
 - ChordReduce handled churn during execution.

2015-07-10
DHT Distributed Computing
Completed Work
ChordReduce
Experimental Results

Experimental Results

Figure: The ratio of hits to generated points $\pi/4$ and a histogram of the ratio of hits to generated points.

- Diminishing returns.
- The 10^7 job was dominated by the overhead communication costs.

2015-07-10
DHT Distributed Computing
Completed Work
ChordReduce
Churn Results

Churn Results

Churn Rate per Second	Execution Time (s)	Execution Time (s) with Churn
0.0025%	120.00	120.00
0.005%	120.00	120.00
0.01%	120.00	120.00
0.025%	120.00	120.00
0.05%	120.00	120.00
0.1%	120.00	120.00
0.25%	120.00	120.00
0.5%	120.00	120.00
0.8%	120.00	120.00

Figure: The results of calculating π by generating 10^7 samples under Churn. Churn is the ratio of nodes that die to the total number of nodes. The high rate of churn is from joining nodes requiring more time.

- We tested at rates from 0.0025% to 0.8% **per second**, 120 times the fastest rate used to test P2P-MapReduce.
- ChordReduce finished twice as fast under the unrealistic levels churn (0.8% per second) than no churn (Table 1).
- Churn is a disruptive force; how can it be aiding the network? We have two hypotheses
- Deleting nodes motivates other nodes to work harder to avoid deletion (a "beatings will continue until morale improves" situation).
- Our high rate of churn was dynamically load-balancing the network. How?
- Nodes that die and rejoin are more likely to join a region owned by a node with larger region and therefore more work.
- It appears even the smallest effort of trying to dynamically load balance, such as rebooting random nodes to new locations, has benefits for runtime. Our method is a poor approximation of dynamic load-balancing, and it still shows improvement.

2015-07-10
DHT Distributed Computing
Completed Work
ChordReduce
Conclusions

Conclusions

Our experiments established:

- ChordReduce can operate under high rates of churn.
- Execution follows the desired topology.
- Scaling occurs in sufficiently large problem sizes.

This makes ChordReduce an excellent platform for distributed and concurrent programming in cloud and loosely coupled environments.

One of the goals coming out of this is that I want to be able to harness this speedup due to churn. I've come up with a number of potential strategies I've listed in the proposal, but a number of them involve nodes being able to create virtual nodes, in other words, be in multiples places at once and have multiple identities.

It turns out the security world has something analogous to that.

2015-07-10

DHT Distributed Computing

Completed Work

Sybil Attack Analysis

The Sybil Attack

The Sybil Attack

- The Sybil attack is a type of attack against a distributed system such as DHT.
- The adversary pretends to be more than one identity in the network.
- Goal of these false identities, which a Sybil is treated as a full member of the network.
- The overall goal is to occlude healthy nodes from one another.
- The Sybil attack is extremely well known, but there is little literature written from the attacker's perspective.

- DHTs don't have a centralized point of failure, but this opens up different strategies for attack
- How the identities are obtained is a question for later, but we assume they don't have to be real hardware.
- By occlude, we mean that that traffic must travel thru the adversary.
- What distinguishes the Sybil from the Eclipse attack is the fact that the Sybil attack relies only on false identity

2015-07-10

DHT Distributed Computing

Completed Work

Sybil Attack Analysis

Analysis

Analysis

The probability you can reach a region between two adjacent nodes in a size n network is:

$$P = \frac{1}{n} \times \text{num_edges} \times \text{num_nodes} \quad (1)$$

An attacker can compromise a portion $P_{\text{num_edges}}$ of the network given by:

$$P_{\text{num_edges}} = \frac{\text{num_edges}}{\text{num_edges} + \text{num_nodes} \times P} \quad (2)$$

- Have proofs!

2015-07-10

DHT Distributed Computing

Completed Work

Sybil Attack Analysis

Figure 3: Our simulation results. The x-axis corresponds to the number of IP addresses the adversary can bring to bear. The y-axis is the probability that a random region between two adjacent normal members of the network can be occluded. Each line maps to a different network size of n .

The x-axis corresponds to the number of IP addresses the adversary can bring to bear. The y-axis is the probability that a random region between two adjacent normal members of the network can be occluded. Each line maps to a different network size of n .

2015-07-10

DHT Distributed Computing

Completed Work

Sybil Attack Analysis

Conclusion

Conclusion

- Our analysis showed an adversary with limited resources can occlude the majority of the paths between nodes.
- An attack of this sort on Malware DHT would cost about \$40,000 USD per hour.
- Moreover, we demonstrated that creating virtual nodes is cheap and easy.

- By cost, that is the Amazon EC2 cost and assumed half the links of 20,000,000 nodes we occluded.

2015-07-10
DHT Distributed Computing
Proposed Work
UrDHT
UrDHT

UrDHT

- UrDHT is a completely abstracted DHT that will serve as a framework for creating DHTs.
- The goal is not only to create a DHT, but to create an easily extensible abstract framework for DHTs.
- Continuation of the work in UrDHT.

- *Ur* as in the germanic prefix for proto or first
- UrDHT is a project that presents a minimal and extensible implementation for all the essential components for a DHT: the different aspects for a protocol definition, the storage of values, and the networking components. Every DHT has the same components, but there has yet to be an all-encompassing framework that clearly demonstrates this.

2015-07-10
DHT Distributed Computing
Proposed Work
UrDHT
UrDHT

UrDHT

- We will be creating a mathematical description of what a DHT is.
- We will implement various DHTs using UrDHT and compare their performance.

- The purpose of doing this is that
- it's cool
- We want a framework for creating DHTs and DHT based applications to be easily available
- I need it to do the rest of my proposal and Brendan needs it for the same reason.

2015-07-10
DHT Distributed Computing
Proposed Work
DHT Distributed Computing
DHT Distributed Computing

DHT Distributed Computing

- We will use UrDHT to implement a few of the more popular DHTs.
- We will see if there is a difference in distributed computing.
- Using UrDHT for all the implementations will enhance the abstract framework for DHTs.


- Additionally this will serve as an example of how to implement our framework.


Introduction
Background
Completed Work
Proposed Work

Autonomous Load-Balancing

- Virtual hadoop.
<http://wiki.apache.org/hadoop/Virtual>
- Bram Cohen.
Incentives build robustness in bittorrent.
In Workshop on Economics of Peer-to-Peer systems, volume 6, pages 68–72, 2003.
- Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica.
Wide-Area Cooperative Storage with CFS.
ACM SIGOPS Operating Systems Review, 35(5):202–215, 2001.
- Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola.
Parameter server for distributed machine learning.
- Andrew Loewenstern and Arvid Norberg.
BEP 5: DHT Protocol.

http://www.bittorrent.org/beps/bep_0005.html,
March 2013.

 Gabriel Mateescu, Wolfgang Gentzsch, and Calvin J. Ribbens.
Hybrid computing—where {HPC} meets grid and cloud
computing.
Future Generation Computer Systems, 27(5):440 – 453, 2011.

 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., 31:149–160, August 2001.