Proposal Defense
Towards a Framework for DHT Distributed Computing

Andrew Rosen

Georgia State University

July 15th, 2015

Table of Contents

Objective

Our objective is to create a generalized framework for distributed computing using Distributed Hash Tables.

Objective

Our objective is to create a generalized framework for distributed computing using Distributed Hash Tables.

Or

Objective

Our objective is to create a generalized framework for distributed computing using Distributed Hash Tables.

Or

We want to build a completely decentralized distributed computing framework.

What do I Mean by Distributed Computing?

A system where we can take a task and break it down into multiple parts, where each part is worked upon individually.

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.

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.

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 we add more machines, we need to be able to handle the increased risk of hardware failure.

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 we add more machines, we 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.

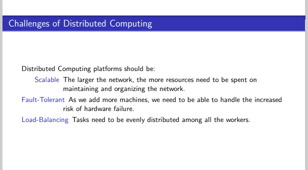
Challenges of Distributed Computing

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 we add more machines, we need to be able to handle the increased risk of hardware failure.

Load-Balancing 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.

Distributed Key/Value Stores

Distributed Hash Tables are mechanisms for storing values associated with certain keys.

- Values, such as filenames, data, or IP/port combinations are associated with keys.
- These keys are generated by taking the hash of the value.
- We can get the value for a certain key by asking any node in the network.

Current Applications

Applications that use or incorporate DHTs:

- P2P File Sharing applications, such as Bittorrent [1] [4].
- Distributed File Storage [2].
- Distributed Machine Learning [3].
- Name resolution in a large distributed database [5].

Georgia State University

How Does It Work?

We'll explain in greater detail later, but briefly:

- DHTs organize a set of nodes, each identified by an ID (their key).
- Nodes are responsible for the keys that are closest it their IDs.
- Nodes maintain a list of other peers in the network.
  - Typically a size $\log(n)$ subset of all nodes in the network.
- Each node uses a very simple routing algorithm to find a node responsible for any given key.

How Does It Work?

We'll explain in greater detail later, but briefly:

- DHTs organize a set of nodes, each identified by an ID (their key).
- Nodes are responsible for the keys that are closest it their IDs.
- Nodes maintain a list of other peers in the network.
  - Typically a size $\log(n)$ subset of all nodes in the network.
- Each node uses 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.

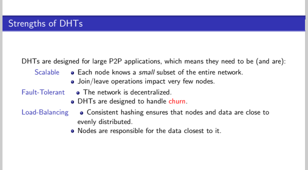
Strengths of DHTs

DHTs are designed for large P2P applications, which means they need to be (and are):

Scalable
- Each node knows a *small* subset of the entire network.
- Join/leave operations impact very few nodes.

Fault-Tolerant
- The network is decentralized.
- DHTs are designed to handle churn.

Load-Balancing
- 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

DHTs Address the Specified Challenges

The big issues in distributed computing can be solved by the mechanisms provided by Distributed Hash Tables.

Uses For DHT Distributed Computing

- Embarrassingly Parallel Computations
  - Brute force cryptography.
  - Genetic algorithms.
  - Markov chain Monte Carlo methods.
  - Any problem that can be framed using Map and Reduce.
- Can be used in either a P2P context or a more traditional deployment.

Uses For DHT Distributed Computing

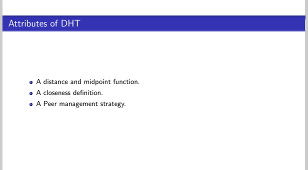
- Embarrassingly Parallel Computations
  - Brute force cryptography.
  - Genetic algorithms.
  - Markov chain Monte Carlo methods.
  - Any problem that can be framed using Map and Reduce.
- Can be used in either a P2P context or a more traditional deployment.

- Need notes here
- Define Monte-Carlo Markov Chain

Table of Contents

Attributes of DHT

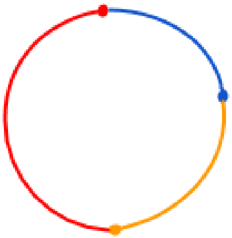
- A distance and midpoint function.
- A closeness definition.
- A Peer management strategy.



- 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

Chord's Closest Metric.

Figure: A Voronoi diagram for a Chord network, using Chord's definition of closest.



Chord Using A Different Closest Metric

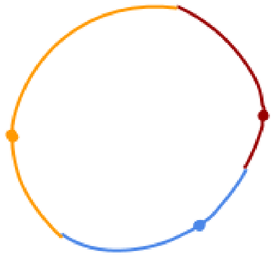
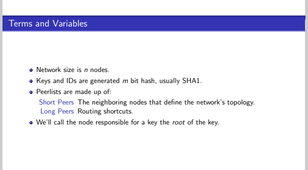


Figure: A Voronoi diagram for a Chord network, where closest if defined by the node being the closest in either direction.

Terms and Variables

- Network size is n nodes.
- Keys and IDs are generated m bit hash, usually SHA1.
- Peerlists are made up of:
  - Short Peers The neighboring nodes that define the network's topology.
  - Long Peers Routing shortcuts.
- 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 gradulally and are not actively pinged.
- We use root as it's is a topology agnostic term.

Functions

lookup(key) Finds the node responsible for a given key.

put(key, $value$) Stores *value* at the node responsible for *key*, where $key = hash(value)$.

get(key) Returns the *value* associated with *key*.

Functions

lookup(*key*) Finds the node responsible for a given key.

put(*key*, *value*) Stores *value* at the node responsible for *key*, where $key = hash(value)$.

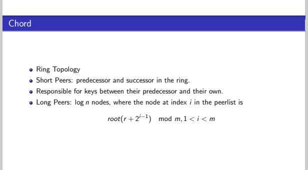
get(*key*) Returns the *value* associated with *key*.

- 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*

Chord

- Ring Topology
- Short Peers: predecessor and successor in the ring.
- Responsible for keys between their predecessor and their own.
- Long Peers: $\log n$ nodes, where the node at index i in the peerlist is

$$root(r + 2^{i-1}) \mod m, 1 < i < m$$



- 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.

Fault Tolerence in Chord

- Local thread maintenance gradually fixes the network topology.
  - Each node "notifies" its successor.
  - The successor replies with a better successor of one exists.
- The long peers are gradually updated performing lookup on each entry.
- Some implementations use predecessor and successor lists.

Handleing Churn in General

- Short peers, the neighbors, are periodically queried to:
  - See of the node is still alive.
  - See if the neighbor knows about better nodes.
- Long peer failures are reacted to and replaced by regular maintenance.

Other DHTs

Table of Contents

ChordReduce

content

DHT and Voronoi Relationship

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

Voronoi Tesselation

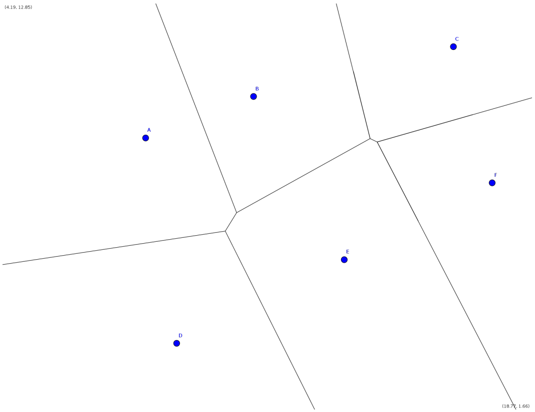


Figure: A set of points and the generated Voronoi regions

Voronoi Tesselation

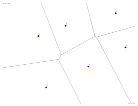


Figure: A set of points and the generated Voronoi regions

Define

- Voronoi generators
- Voronoi Region
- Voronoi Tessellation/ Diagram

Delaunay Triangulation

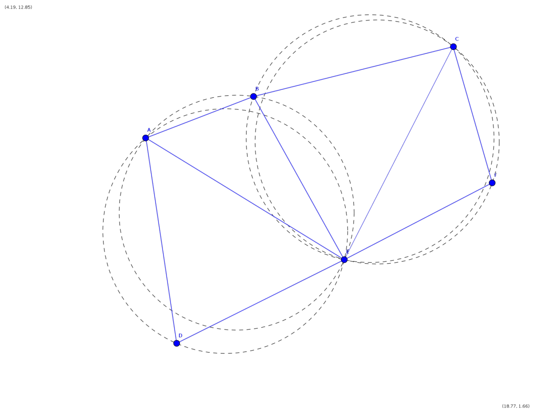


Figure: The same set of nodes with their corresponding Delaunay Triangulation.

DHT and Voronoi Relationship cont'd

- Coprime

VHash

- Voronoi-based Distributed Hash Table

DGVH

content

Sybil Analysis

content

Table of Contents

UrDHT

This kind of framework does not exist.

UrDHT

This kind of framework does not exist.

At Brendan's suggestion We plan on extending the functionality of DHTs, post poll

DHT Distributed Computing

content

Autonomous Load-Balancing

content

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.