Distributed Hash Tables

Pastry

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

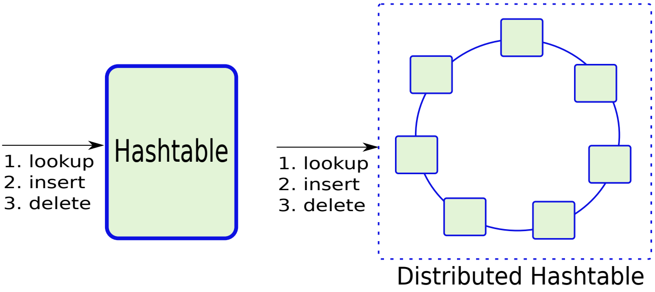
Normal Hashtables

- Hashtable: Contains a set of key-value pairs. If the user supplies the key, the hashtable returns the value.
- Basic operations.

insert(key,value)	Inserts the key,value pair into the hashtable.
lookup(key)	Returns the value, or null if there is no value.
delete(key)	Deletes the key
Time Complexity	Approximately, $\theta(1)$

- Need a sophisticated hash function to map keys to unique locations.
- Need to resolve collisions through chaining or rehashing.

Distributed Hashtables(DHT)



Salient Points of DHTs

- They can store more data than centralized databases.
- DHTs are the only feasible options for web-scale data: Facebook, LinkedIn, Google
  - Assume that a bank has 10 crore customers (0.1 billion)
  - Each customer requires storage equivalent to the size of this latex file.
  - Total storage requirement: 8 KB $\times$ 0.1 billion = 0.8 TB

Salient Points of DHTs

- They can store more data than centralized databases.
- DHTs are the only feasible options for web-scale data: Facebook, LinkedIn, Google
  - Assume that a bank has 10 crore customers (0.1 billion)
  - Each customer requires storage equivalent to the size of this latex file.
  - Total storage requirement: 8 KB $\times$ 0.1 billion = 0.8 TB

- A user is sharing 100 songs : 500 MB/user
- There are 10 crore(0.1 billion) users
- Storage: 50 PB (petabytes)

Salient Points of DHTs

- They can store more data than centralized databases.
- DHTs are the only feasible options for web-scale data: Facebook, LinkedIn, Google
  - Assume that a bank has 10 crore customers (0.1 billion)
  - Each customer requires storage equivalent to the size of this latex file.
  - Total storage requirement: 8 KB $\times$ 0.1 billion = 0.8 TB

- A user is sharing 100 songs : 500 MB/user
- There are 10 crore(0.1 billion) users
- Storage: 50 PB (petabytes)

There is a difference of an order of magnitude !!!

Advantages of DHTs

- DHTs scale, and are ideal candidates for web scale storage.
- They are more immune to node failures. They use extensive data replication.
- DHTs also scale in terms of the number of users. Different users are redirected to different nodes based on their keys. (better load balancing).
- In the case of Torrent applications: they reduce the legal liability since there is no dedicated central server. ☺

Major Proposals

Pastry, Chord, Tapestry, CAN, Fawn

Outline

Salient Points

- Scalable distributed object location service.
- Uses a ring based overlay network.
- The overlay network takes into account network locality.
- Automatically adapts to the arrival, departure, and failure or nodes.
- Pastry has been used as the substrate to make large storage services (PAST) and scalable publish/subscribe system(SCRIBE).
  - PAST is a large scale file storage service.
  - SCRIBE stores a massive number of topics in the DHT. When a topic changes, the list of subscribers are notified.

Design of Pastry

- Node $\rightarrow$ has a unique 128 bit nodeId.
- The nodes are conceptually organized as a ring, arranged in ascending order of nodeIds.
- nodeIds are generated by computing a cryptographic hash of the node's IP address or public key.
- Basic idea of routing:
  - Given a key, find its 128 bit hash.
  - Find the node whose nodeId is numerically closest to the hash-value.
  - Send the request to that node.

Advantages of Pastry

- Pastry can route a request to the right node in less than $\lceil log_{2^b}(N) \rceil$ steps. b is typically 4.
- Eventual delivery is guaranteed unless $L/2$ nodes with adjacent nodeIds fail simultaneously. L = 16 or 32.
- Both nodeIds and keys are thought to be base 2^b numbers. If we assume that $b = 4$, then these are hexadecimal numbers.

Basic Operation

- In each step, the request is forwarded to a node whose shared prefix with the key is at least 1 digit (b bits) more than the length of the shared prefix with the current node's nodeId.
- If no such node is found, then the request is forwarded to a node, which has the same length of the prefix but is numerically closer to the key.

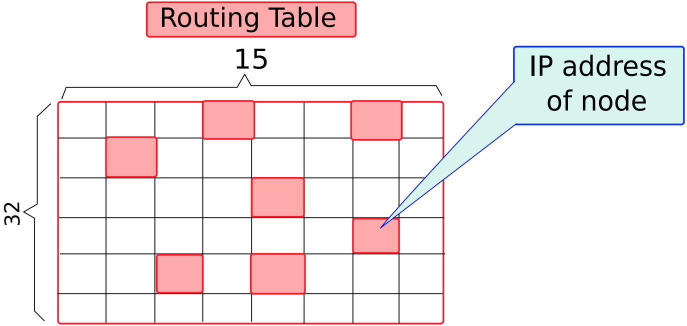
Outline

Pastry Node

Structure of a Pastry Node

It contains a routing table, neighborhood table, and a leaf set

Structure of the routing table



Routing Table ($\mathcal{R}$)

- The routing table contains 32 rows ($1 \ldots 32$).
- The entries at row i (count starts from 1) point to nodes that share the first $(i-1)$ digits of the prefix with the key.
- Each row contains $2^b - 1$ columns.
- Each cell refers to a base 2^b digit.
- If the digit associated with a cell matches the i^{th} digit of the key, then we have a node that matches the key with a longer prefix.
- We should route the request to that node.

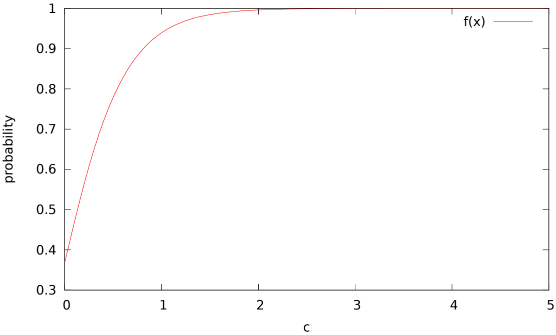
Some Maths

- The probability that two hashes have the first m digits common is 16^{-m}. Let us assume $2^b = 16$.
- The probability that a key does not have its first m digits matching with the first m digits of a node: $1 - 16^{-m}$
- The probability that the key does not have a prefix match of length m with all of the nodes: $(1 - 16^{-m})^n$
- Let $m = c + log_{16}(n)$
- We have:

$$\begin{aligned} prob = (1 - 16^{-m})^n &= (1 - 16^{-c-log_{16}(n)})^n \\ &= (1 - 16^{-c}/n)^n \quad (\lambda = 16^{-c}) \\ &= \left(1 - \lambda/n)^{n/\lambda}\right)^{\lambda} \\ &= e^{-\lambda} \quad (n \rightarrow \infty) \end{aligned}$$

Some Maths-II

- As, c becomes larger (let's say 5), $\lambda = 16^{-c}$ becomes very small. Example: $16^{-5} = 9.5 * 10^{-7}$.
- Essentially: $\lambda \rightarrow 0$



Neighborhood Set and Leaf Set

Neighborhood Set ($\mathcal{M}$)

- It contains M nodes that are closest to the node according to a proximity metric.
- Typically contains 2^{b+1} entries.

Leaf Set ($\mathcal{L}$)

- Contains $L/2$ nodes with a numerically closest larger nodeIds.
- Contains $L/2$ nodes with a numerically closest smaller nodeIds.
- Typically 2^b.

Routing Algorithm

Algorithm 1: Routing algorithm

Input: key K, routing table $\mathcal{R}$, Hash of the key $\rightarrow D$
Ouput: Value V

if $\mathcal{L}_{-L/2} \leq D \leq \mathcal{L}_{L/2}$ **then**
    /* K is within the range of the leaf set */
    forward K to L_i such that $| L_i - K |$ is minimal
end
else
    $l \leftarrow$ common_prefix(K, *nodeId*)
    if $\mathcal{R}(l+1, D_{l+1}) \neq$ *null* **then**
        forward to $\mathcal{R}(l+1, D_{l+1})$
    end
    else
        forward to ($T \in \mathcal{L} \cup \mathcal{R} \cup \mathcal{M}$) such that
        prefix(T, K) $\geq l$
        $| T - K | < | nodeId - K |$
    end
end

Explanation

- The node first checks to find if the key is within the leaf set. If so, it forwards the messages to the closest node (by nodeId) in the leaf set.
- Otherwise, Pastry forwards the message to a node with one more matching digit in the common prefix.
- In the rare case, when we are not able to find a node that matches the first two criteria, we forward the request to any node that is closer to the key than the current nodeId. Note that it still needs to have a match of at least l digits.

Performance and Reliability

- If $L/2$ nodes in the leaf set are alive then the message can always be passed on to some other node.
- At every step, we are (with high probability) either searching in the leaf set or moving to another node.
- If the key is within the range of the leaf set $\rightarrow$ it is at most one hop away
- Otherwise, at every step we increase the length of the matched prefix by 2^b.

Routing Time Complexity

The average routing time is thus $O(log_{2^b}(N))$.

Outline

Node Arrival

- Assume that node X wants to join the network.
- It locates another nearby node A.
- A can also be found with expanding ring multicast.
- A forwards the request to the numerically closest node – Z.
- Nodes, A, Z, and all the nodes in the path from A to Z send their routing tables to X.
- X uses all of this information to initialize its tables.

Table Initialization

Neighborhood Set

Since A is close to X. X copies A's neighborhood set.

Leaf Set

Z is the closest to X (nodeId). X uses Zs leaf set to form its own leaf set.

Table Initialization - II

Routing Table

- Assume that A and X do not share any digits in the prefix (General Case).
- The first row of the routing table is independent of the nodeId (No Match). X can $A's$ row to initialize its first row.
- Every node in the path from A to Z has one additional digit matching with X. Let B_i be the i^{th} entry in the path from A to Z.
- Observation: B_i shares i digits of its prefix with X. Use its $(i+1)^{th}$ row in its routing table to populate the $(i+1)^{th}$ row of the routing table of X.
- Finally X transmits its state to all the nodes in $\mathcal{L} \cup \mathcal{R} \cup \mathcal{M}$.

Node Departure – Leaf Set

- Nodes might just fail or leave without notifying.

Repairing the leaf set

- Assume that a leaf $\mathcal{L}_{-k}$ fails. $(-L/2 < k < 0)$.
- In this case, the node contacts $\mathcal{L}_{-L/2}$.
- It gets its leaf set and merges it with its leaf set.
- For any new nodes added, it verifies their existence by pinging them.

Node Departure – Routing Table and Neighborhood Set

Repairing the Routing Table

- Assume that $\mathcal{R}(l, d)$ fails.
- Try to get a replacement for it by contacting $\mathcal{R}(l, d')(d \neq d')$
- If it is not able to find a candidate, it casts a wider net by asking $\mathcal{R}(l+1, d')(d \neq d')$

Repairing the Neighborhood Set ($\mathcal{M}$)

- A node periodically pings its neighbors.
- If a neighbor is found to be dead, it gets $\mathcal{M}$ from its neighbors and repairs its state.

Maintaining Locality

- Assume that before node X is added, there is good locality.
  - All the nodes in the routing table point to nearby nodes.
- We add a new node X
  - We start with a nearby node A and move towards Z(closest by nodeId).
  - Let B_i be the i^{th} node in the path. (Induction assumption: B_i has locality)
  - B_i is fairly close to X because it is fairly close to A.
  - Since we get the i^{th} row of the routing table from B_i, and B_i has locality, the i^{th} row of the routing table of X.
  - Thus, X has locality.

Maintaining Locality

- Assume that before node X is added, there is good locality.
    - All the nodes in the routing table point to nearby nodes.
- We add a new node X
    - We start with a nearby node A and move towards Z(closest by nodeId).
    - Let B_i be the i^{th} node in the path. (Induction assumption: B_i has locality)
    - B_i is fairly close to X because it is fairly close to A.
    - Since we get the i^{th} row of the routing table from B_i, and B_i has locality, the i^{th} row of the routing table of X.
    - Thus, X has locality.

Induction hypothesis proved

Tolerating Byzantine Failures

- Have multiple entries in each cell of the routing table.
- Randomize the routing strategy.
- Periodically send IP broadcasts and multicasts (expanding ring) to connect disconnected networks.

Outline

Setup

- 100,000 nodes
- Each node runs a Java based VM
- Each node is assigned a co-ordinate in an Euclidean plane.

Performance Results

- The average number of hops varies linearly with the number of nodes (in the log scale).
- 2.5 hops for 1000 nodes. 4 hops 100,000 nodes.
- For 100,000 nodes and 200,000 lookups the probability distribution is as follows.
  - 2 hops - 1.5%
  - 3 hops - 16.4%
  - 4 hops - 64%
  - 5 hops - 17%
- With a complete routing table, the hop count would have been 30% lower.

Source [1]

Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems by Antony Rowstron and Peter Druschel in Middleware 2001