Lesson 5

A DISTRIBUTED HASH TABLE: CHORD FINGER TABLES AND ROUTING

Laura Ricci
6/3/2017
A Distributed Hash Table: Chord
Laura Ricci

Dipartimento di Informatica
Università degli Studi di Pisa

LESSON OUTLINE

- DHT: recap
- Chord: general features
- The overlay topology
- Routing
  - Modelling routing through Markov chains
- Self organization
  - node join
  - voluntary leave
  - faults
- Analysis of the Chord topology through complex networks analysis tools
CHORD: INTRODUCTION

Paper reference


developed in 2001 from a research group formed by researcher from MIT, University of California
CHORD: INTRODUCTION

- Based on a few, simple but powerful concepts:
  - easily to define and to implement
  - elegance
  - possibility of defining optimizations
- Main characteristics:
  - Routing (to implement \texttt{FindPeer}, to find a peer managing a bucket)
    - a flat logical address space: peer addresses are \textit{m-bits identifiers}, instead of IP addresses
    - efficient routing: \textit{log(N) hops with high probability}, where \(N\) is the total number of peers in the system
    - routing table size: \textit{log(N) with high probability}
  - self-organization
  - self-adaptation in presence of new nodes joins and voluntary/abrupt nodes leave.
CHORD: GENERAL CONCEPTS

- Each node (host) is paired with an identifier $id$, obtained by SHA-1
  
  \[ id_{node} = SHA-1 (IP \text{ address, port}) \]

- Each data is paired with a unique identifier $k$ (or key), obtained by SHA-1
  
  \[ k = SHA-1 (data) \]

- Keys and nodes are mapped onto the same logical address space

- Hash-table interface
  - put (key, value) to insert data
  - value = get (key) to look up data
**CHORD: THE OVERLAY TOPOLOGY**

- Hypothesis: consider identifiers (returned by SHA) of m bits, $[0, 2^m-1]$.

- Define an ordering between the identifiers, based on their numerical value such that successor of $2^m-1$ is 0 (modulo operations).

- The ordering may be represented by a ring (Chord Ring).

- Mapping Algorithm

  An identifier $K$ is assigned to the first node $n$ of the ring whose identifier is greater or equal (modulo the size of the ring) to $K$.

  $$n(K) = \text{successor}(K)$$

- $n$ is the first node detected starting from $K$ and following the Chord ring clockwise.
THE CHORD RING

- the ring includes identifiers, the arithmetic is mod $2^{160}$ (in the example mod $2^3$)
- the key and the corresponding value are managed by the successor of the key in the **clockwise ordering**
THE CHORD RING

- the ring includes identifiers, the arithmetic is mod $2^{160}$ (in the example mod $2^3$)
- the key and the corresponding value are managed by the successor of the key in the clockwise ordering
• the ring includes identifiers, the arithmetic is mod $2^{160}$ (in the example mod $2^3$)
• the key and the corresponding value are managed by the successor of the key in the clockwise ordering
- The ring includes identifiers, the arithmetic is mod $2^{160}$ (in the example mod $2^{3}$).
- The key and the corresponding value are managed by the successor of the key in the clockwise ordering.
THE CHORD RING

- the ring includes identifiers, i.e., the arithmetic is mod \(2^{160}\) (in the example mod \(2^3\))
- the key and the corresponding value are managed by the successor of the key in the clockwise ordering

The Chord Ring includes:
- identifiers
- key

For example:
- successor(1) = 1
- successor(2) = 3
- successor(6) = 0

In the diagram:
- Node 6 is predecessor of node 0
- Node 2 is predecessor of node 3
- Node X represents the key managed by its successor.
• How routing is implemented?

• It depends from the information stored in the routing table

• **Minimal Routing Table** of a node:
  - stores only the successor link
  - this is the only knowledge of a node about other nodes of the ring
A simple routing algorithm:
- the routing table of each node has a unique link towards the clockwise successor on the ring
- send the query with key=x to the successor until n =successor(x) is detected
- n returns the query results

Advantages:
- simple
- routing tables $O(1)$

Disadvantages:
- routing $O(\frac{1}{2} \times n)$, linear
- a single node crash breaks the ring
A simple routing algorithm, each node:
- has a single link towards its successor
- sends the key=x to its successor until it finds it does not find
  \( n = \text{successor}(x) \)
- \( n \) returns the query results

Advantages:
- simple
- routing table \( O(1) \)

Disadvantages:
- Routing \( O(\frac{1}{2} \times n) \), linear
- a single node crash breaks the ring
A simple routing algorithm, each node:

- has a single link towards its successor
- sends the key=x to its successor until it finds it does not find \( n = \text{successor}(x) \)
- \( n \) returns the query results

Advantages:

- simple
- routing table \( O(1) \)

Disadvantages:

- routing \( O(\frac{1}{2} * n) \), linear
- a single node crash breaks the ring
A simple routing algorithm, each node:
- has a single link towards its successor
- sends the key=x to its successor until it finds it does not find
  \( n = \text{successor}(x) \)
- \( n \) returns the query results

Advantages:
- simple
- routing table \( O(1) \)

Disadvantages:
- Routing \( O\left(\frac{1}{2} \times n\right) \), linear
- a single node crash breaks the ring
A distributed algorithm: use Remote Procedure Calls (RPC) to code it

- \( (a, b] \) the segment of the ring moving clockwise from but not including \( a \) until and including \( b \).
- \( n.\text{foo}(\ ) \) denotes an RPC of \( \text{foo}(\ ) \)@node \( n \).
- \( n.\text{bar} \) denotes and RPC to fetch the value of the variable bar in node \( n \).
PUT and GET are nothing but lookups!
CHORD ROUTING: REDUCING NUMBER OF STEPS

- Each node has links towards $z$ neighbours.

- If $z = n$, the overlay is a complete mesh.
  - Routing hops: $O(1)$
  - Size of the routing tables: $O(n)$, limited scalability.

- A compromise: Each node stores several links towards some close neighbours (on the ring) and a few links towards far neighbours.
  - A limited number of links for each node.
  - Routing is more accurate in the neighbourhood of a node, more approximate towards far neighbours.
  - Routing algorithm:
    - Send a query for the key $k$ to the farthest known predecessor of $k$. 
CHORD ROUTING: REDUCING THE COMPLEXITY

Let us define the distance between two identifier $I_1$ and $I_2$ of the Chord ring as the number of identifier between $I_1$ and $I_2$

Basic idea for building the routing table (finger table)

- the node $n$ with identifier $x$ in the ring knows a set of at most $m$ nodes (for a ring of $m$ bits) and the distance from these nodes exponentially increases
  - Chord considers nodes at distance approximately $2^i$ from $x$, with $0 \leq i \leq m-1$
  - the number of nodes known by a node $n$ generally is less than $m$

- furthermore, each node knows its successor and its predecessor on the ring
FINGER TABLES

A Distributed Hash Table: Chord
Laura Ricci
Each node owns a **finger table** (routing table)

- if $m$ is the number of bits exploited for the identifiers, the table includes at most $m$ links to Chord nodes
- at node $n$: the entry $\text{finger}[i]$ is a pointer to $\text{successor}(n + 2^{i-1})$, $1 \leq i \leq m$

**Data structure of Node 0**

<table>
<thead>
<tr>
<th>i</th>
<th>target</th>
<th>link.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Each node owns a finger table (routing table)
- if $m$ is the number of bits exploited for the identifiers, the table includes at most $m$ links to Chord nodes
- at node $n$: the entry $\text{finger}[i]$ is a pointer to $\text{successor}(n + 2^{i-1})$, $1 \leq i \leq m$

Data Structure of Node 0

<table>
<thead>
<tr>
<th>finger table</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>target</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Data Structure of Node 1

<table>
<thead>
<tr>
<th>finger table</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>target</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Each node owns a **finger table** (routing table)

- if $m$ is the number of bits exploited for the identifiers, the table includes at most $m$ links to **Chord** nodes
- at node $n$: the entry $\text{finger}[i]$ is a pointer to $\text{successor}(n + 2^{i-1})$, $1 \leq i \leq m$

Data Structure of Node 0

<table>
<thead>
<tr>
<th>finger table</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>target</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Data Structure of Node 1

<table>
<thead>
<tr>
<th>finger table</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>target</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Data Structure of Node 3

<table>
<thead>
<tr>
<th>finger table</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>target</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
DATA STRUCTURES OF A NODE

- Each node maintains some data structures supporting routing

1) the finger table:
   - has at most m entries (for instance m=160)
   - but....the same value is often present in more than one entry
   - actually, the finger table has, on the average, a logarithmic number of entries with respect to the number of nodes of the overlay

2) a link to the successor and to the predecessor on the ring

3) the keys mapped to that node

4) a set of pointers to some successor nodes of the ring to guarantee the network consistency for dynamic nodes join/leave.

Chord Overlay:

a logarithmic mesh of the nodes of the ring
Routing Protocol: each node $n$ propagates a query with key $k$ to the farthest finger preceding $k$, by considering the clockwise ordering.

The propagation of the key goes on until the node $n$ such that:

$$n < k \text{ and } \text{successor}(n) \geq k \text{ (arithmetic modulo } 2^m)$$

in this case the successor($n$) owns the key.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$2^i-1$</th>
<th>Target</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>20</td>
<td>23</td>
</tr>
</tbody>
</table>
• Routing Protocol: each node $n$ propagates the query with key $k$ to the farthest finger preceding $k$, by considering the clockwise ordering.
• The propagation of the key goes on until the node $n$ such that:
  $n < k$ and $\text{successor}(n) \geq k$ (arithmetic modulo $2^m$)

in this case the successor($n$) owns the key.

### Lookup (44)

<table>
<thead>
<tr>
<th>$i$</th>
<th>$2^i - 1$</th>
<th>Target</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>20</td>
<td>23</td>
</tr>
</tbody>
</table>
DHT LOOK UP

- **Routing Protocol**: each node \( n \) propagates the query with key \( k \) to the farthest finger preceding \( k \), by considering the clockwise ordering.
- The propagation of the key goes on until the node \( n \) such that:
  \[
  n < k \text{ and } \text{successor}(n) \geq k \text{ (arithmetic modulo } 2^m)\]
  in this case the successor\( (n) \) owns the key.

### Table: DHT Look Up

<table>
<thead>
<tr>
<th>i</th>
<th>( 2^{i-1} )</th>
<th>Target</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>31</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>55</td>
<td>56</td>
</tr>
</tbody>
</table>

**lookup (44)**
- **Routing Protocol**: each node $n$ propagates the query with key $k$ to the farthest finger preceding $k$, by considering the clockwise ordering.
- The propagation of the key goes on until the node $n$ such that:
  $$n < k \text{ and } \text{successor}(n) \geq k \text{ (arithmetic modulo } 2^m)$$
in this case the successor($n$) owns the key.

### Table: Lookup (44)

<table>
<thead>
<tr>
<th>$i$</th>
<th>$2^i - 1$</th>
<th>Target</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>31</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>55</td>
<td>56</td>
</tr>
</tbody>
</table>
• **Routing Protocol**: each node $n$ propagates the query with key $k$ to the farthest finger preceding $k$, by considering the clockwise ordering.
• The propagation of the key goes on until the node $n$ such that:
  
  $n < k$ and $\text{successor}(n) \geq k$ (arithmetic modulo $2^m$)

  in this case the $\text{successor}(n)$ owns the key.

<table>
<thead>
<tr>
<th>$2^i - 1$</th>
<th>Target</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>7</td>
</tr>
</tbody>
</table>
DHT LOOK UP

- **Routing Protocol**: each node \( n \) propagates the query with key \( k \) to the farthest finger preceding \( k \), by considering the clockwise ordering.

- The propagation of the key goes on until the node \( n \) such that:
  \[
  n < k \text{ and } \text{successor}(n) \geq k \text{ (arithmetic modulo } 2^m)\]

in this case the successor\((n)\) owns the key.

---

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 2^{i-1} )</th>
<th>Target</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>40</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>41</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>43</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>47</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>55</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

**lookup** (44)
• **Routing Protocol**: each node $n$ propagates the query with key $k$ to the farthest finger preceding $k$, by considering the clockwise ordering.

• The propagation of the key goes on until the node $n$ such that:
  
  $$n < k \text{ and } \text{successor}(n) \geq k \text{ (arithmetic modulo } 2^m)$$

in this case the successor($n$) owns the key.

### Lookup (44)

<table>
<thead>
<tr>
<th>$i$</th>
<th>$2^{i-1}$</th>
<th>Target</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>43</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>46</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>50</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>58</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>10</td>
<td>13</td>
</tr>
</tbody>
</table>
DHT LOOK UP

- **Routing Protocol**: each node \( n \) propagates the query with key \( k \) to the farthest finger preceding \( k \), by considering the clockwise ordering.
- The propagation of the key goes on until the node \( n \) such that:
  \[
  n < k \text{ and } \text{successor}(n) \geq k \] (arithmetic modulo \( 2^m \))
  in this case the successor\((n)\) owns the key

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 2^{i-1} )</th>
<th>Target</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>43</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>46</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>50</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>58</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>10</td>
<td>13</td>
</tr>
</tbody>
</table>

\[\text{lookup (44)} = 45\]
// ask node n to find the successor of id
procedure n.findSuccessor(id) {
    if (predecessor ≠ nil and id ∈ (predecessor, n]) then return n
    else if (id ∈ (n, successor]) then
        return successor
    else // forward the query around the circle
        return successor.findSuccessor(id)
}
// ask node n to find the successor of id

procedure n.findSuccessor(id) {
    if (predecessor \neq \text{nil} \text{ and } id \in (\text{predecessor}, n)) \text{ then return } n
    else if (id \in (n, \text{successor})) \text{ then}
        return successor
    else { // forward the query around the circle
        m := closestPrecedingNode(id)
        return m.findSuccessor(id)
    }
}

// search locally for the highest predecessor of id

procedure closestPrecedingNode(id) {
    for i = m downto 1 do {
        if (finger[i] \in (n, id)) \text{ then}
            return finger[i]
    }
    return n
}