Distributed K-Ary System

A seminar presentation

Konstantin Welke welke@fillibach.de

Arne Vater and Prof. Schindelhauer
Professorship for Computer Networks and Telematik
Department of Computer Science
University of Freiburg

2007-03-01
Outline

1. Introduction
2. Network Structure
3. Applications
4. Handling Dynamism
5. Conclusion
1 Introduction

2 Network Structure

3 Applications

4 Handling Dynamism

5 Conclusion
What can DKS do?

**Distributed Hash Table (DHT)**
- store and retrieve key/value-pairs

**Message Passing**
- Broadcast
- Multicast

**Fault Tolerance**
- Protect against failures
- Degree of tolerance is configurable
Why is that interesting?

- DHTs are a research subject
  - First academic papers in 2001
- DKS claims to formalize a class of approaches
  - Chord
  - Pastry
  - Kademlia
- This implies optimization!
Basic structure

- Overlay network
- Three parameters: $N$, $k$, $f$
- $N =$ maximum number of nodes
  - Ring structure
- $k =$ search tree arity
  - contact any node in $\log_k N$ steps
- $f =$ fault tolerance
  - every key stored $f$ times
  - $(f-1)$ nodes may fail
Definitions

**Definition: Modulo arithmetic**
- \( a \oplus b = (a + b) \mod N \)

**Definition: Distance**
- \( d(a, b) = b \ominus a \)
- Clockwise!

**Definition: Successor \( S(x) \)**
- First node starting at \( x \)
- \( S(x) = x \oplus \min \{d(x, y) \mid y \in P\} \)
**Definitions**

**Definition: Next successor `succ`**
- First node after $x$
- $succ = S(x \oplus 1)$

**Definition: Predecessor `pred`**
- First node before $x$
- $\iff$ Last node after $x$
- $pred = x \oplus \max\{d(x, y) \mid y \in P\}$

---

Konstantin Welke welke@fillibach.de  
Distributed K-Ary System
Basic Structure

Every node has a route to its
- next successor
- predecessor

A route is
- overlay address (node number)
- underlay address (TCP/IP)
Addressing problem

How can two nodes communicate?

- Does node x exist?
- What is its underlay address?
Addressing problem

How can two nodes communicate?

- Does node x exist?
- What is its underlay address?
How can two nodes communicate?

- Does node x exist?
- What is its underlay address?
Addressing problem

How can two nodes communicate?

- Does node x exist?
  - Talk to successor

- What is its underlay address?
  - His predecessor knows
  - Pass message around
  - Inefficient!

- We need more routes!

Konstantin Welke welke@fillibach.de

Distributed K-Ary System
How can two nodes communicate?

- Does node x exist?
  - Talk to successor
- What is its underlay address?
  - His predecessor knows
  - Pass message around
  - Inefficient!
- We need more routes!
How can two nodes communicate?

- Does node x exist?
  - Talk to successor
- What is its underlay address?
  - His predecessor knows
  - Pass message around
  - Inefficient!

We need more routes!
Addressing problem

How can two nodes communicate?

- Does node x exist?
  - Talk to successor
- What is its underlay address?
  - His predecessor knows
  - Pass message around
  - Inefficient!
- We need more routes!
Idea: Add more routes

- Idea: Each node divides the network in $k$ intervals
Idea: Add more routes

- **Idea:** Each node divides the network into $k$ intervals

- **Store a pointer to the successor of each interval**
  - But any node in the interval is okay, too

- **Send to closest node**
  - Remember: Distance is clockwise
  - Don’t overshoot
Idea: Add more routes

Definition: Successor of an interval

First node in Interval \([n, m]\):
- if \(S(n) < m\)
  - \(S([n, m]) = S(n)\)
- else
  - \(S([n, m]) = \text{nil}\)
Idea: Add more routes

- So far
  - max. \( \frac{N}{k} \) hops instead of \( N \) hops
- Apply recursively to first interval
  - each hop divides distance by \( k \)
  - \( L = \log_k N \) levels
  - max. \( L \) hops
  - Routing table size \( k \times L \)
- This view is unique for each node (!)
View is unique for each node
N=16, k=2, node 0

Konstantin Welke welke@fillibach.de
Distributed K-Ary System
View is unique for each node
N=16, k=2, node 1
View is unique for each node
N=16, k=2, node 2
View is unique for each node
N=16, k=2, node 3
View is unique for each node
N=16, k=2, node 4
Idea: Add more routes

Example: Send Message from 0 to S(13)

- \( N=16, \ k=2 \)
- Send message from 0 to S(13)
- Node 0 does not know whether 13 exists.
- Send message to closest node in routing table:
  - Node 0 sends to node 10
Example: Send Message from 0 to $S(13)$

- $N=16$, $k=2$
- Send message from 0 to $S(13)$
- Node 0 does not know whether 13 exists.
- Send message to closest node in routing table:
  - Node 0 sends to node 10
  - Node 10 sends to node 11
Example: Send Message from 0 to S(13)

- N=16, k=2
- Send message from 0 to S(13)
- Node 0 does not know whether 13 exists.
- Send message to closest node in routing table:
  - Node 0 sends to node 10
  - Node 10 sends to node 11
  - Node 11 knows node 13 does not exist (as 11.succ = 14)
  - Node 11 sends to S(13) = 14
Example: Send Message from 0 to S(13)

- N=16, k=2
- Send message from 0 to S(13)
- 3 messages (0 to 10, 10 to 11, 11 to 14)
  - \( \leq \log_k N = 4 \)
Any questions so far?
Outline

1. Introduction
2. Network Structure
3. Applications
4. Handling Dynamism
5. Conclusion
Distributed Hash Table

- Store and retrieve key/value-pairs
- Known Hash-function maps key to identifier $H(key) \to [0, N[\]
- Store key/value-pair at node $S(H(key))$
  - “Node n is responsible for key”
- Store/retrieve: send a message to responsible node

Distributed K-Ary System

Konstantin Welke welke@fillibach.de
One DKS can store multiple DHTs.

If a new node joins, it fetches all key/value-pairs it is responsible for from its successor.

Image taken from DKS presentation by Ali Ghodsi, Seif Haridi, Luc Onana at http://dks.sics.se
Broadcast

- Send message to all Level $l = 1$ nodes
- These recursively send to all Level $l + 1$ nodes
- Each node sends max. $\log_k N$ messages
- Reaches $N$ nodes with $N$ Messages
Multicast

- Join multicast group
- Broadcast to group

Image taken from DKS presentation by Ali Ghodsi, Seif Haridi, Luc Onana at http://dks.sics.se
What about Dynamism?

Nodes can

- join
- leave
- fail
Joining node must know at least one existing node

Joining node is assigned identifier \( n \)

Joining node \( n \) contacts \( e = \text{succ}(n) \)

Insertation like in double-linked list

Transfer identifiers in \([p, n]\) to \( n \)
Joining: forwarding and locking

- While n joins, e forwards all messages from p to n (step 4)
  - Lookups stay consistent
  - p tells e when its done
- Locking ensures consistency
  - Each node hosts a lock
  - n locks itself in step 1
  - n tries to lock e in step 2
  - in step 5, both locks are released
Joining: populate routing table

- Next successor e sends approximate routing table to n
  - Any node in a given interval is ok
- Node n can lookup better nodes, if it wants to
- Node n tells all nodes in its routing table that it points to them
  - They store node n in their backlist
Leaving

- Node q wants to leave
- Leaving like in double-linked lists
- Node q starts transferring all identifiers to r
- Node q tells all nodes in its routing table and backlist that it is leaving

Konstantin Welke welke@fillibach.de  Distributed K-Ary System
Leaving: forwarding and locking

- Node q first locks itself, then its successor
  - If \( q > r \): Node q locks its successor, then itself
  - Freedom of Deadlocks, Livelocks
- Node q forwards all messages to its successor
  - Lookup consistency
- Node q releases all locks when done
Correction-on-use

- No periodic stabilization!
- Each time a message is sent, embed level and interval.
- Receiving node can tell if information is correct
  - If not, tell sender
- Network corrects itself
- Needs enough “user traffic”
Correction-on-use

- No periodic stabilization
- Each time a message is sent, embed *level* and *interval*.
- Receiving node can tell if information is correct
  - If not, tell sender
- Network corrects itself
- Needs enough “user traffic”
Correction-on-use

- No periodic stabilization!
- Each time a message is sent, embed *level* and *interval*.
- Receiving node can tell if information is correct
  - If not, tell sender
- Network corrects itself
- Needs enough “user traffic”
Handling Failures

- Each node maintains successor list of length \( f \)
  - quick replacement when next successor fails
- same for predecessor list
Symmetric Replication

- Each information is stored $f$ times
  - At nodes $n \oplus i \frac{N}{f}$, $i \in N_0$
  - All nodes $n \oplus \frac{N}{f}$ are responsible for the same set of identifiers
- Store in DHT: send $f$ messages
  - Problem of race conditions not mentioned anywhere

Konstantin Welke welke@fillibach.de
Outline

1 Introduction
2 Network Structure
3 Applications
4 Handling Dynamism
5 Conclusion
Conclusion

- Routing table size $k \times \log_k N$
- Lookup node with max. $\log_k N$ messages
- Optimal Broadcast with max $\log_k N$ messages per node
- Handles concurrency in joins, leaves
- Problems with concurrency in replication

Ali Ghodsi Distributed k-ary System: Algorithms for Distributed Hash Table KTH — Royal Institute of Technology, 2006 Stockholm, Sweden
Bibliography


IEEE Computer Society Washington, DC, USA
El-Ansary, S.
A Framework For The Understanding, Optimization and Design Of Structured Peer-To-Peer Systems
Ph. D. thesis, Swedish Institute of Computer Science (SICS)
Kista, Sweden, 2003
That’s all folks!

Thank you for your attention!

Any questions?