OM-QoS: Quality of Service for Overlay Multicast using Chord

RVS Seminar HS08, November 26th, 2008
Andreas Rüttimann
Overview

- Chord: QoS, Multicast repetition
- Chord optimizations
  - Why?
  - Examples
- Chord performance using Multicast and protocol based QoS
- End-to-End Delay
  - Goal / Problems
  - Protocol based optimizations
    - Receiver driven Multicast
    - Sender driven Multicast
  - Framework optimizations
- Outlook
Chord Basics

- DHT based, structured P2P network
- Nodes have pointers to k successors
- Finger table: pointers to $\log_2(\text{idspace})$ nodes -> better knowledge about close nodes
- Stabilization
Chord Multicast

> Sender driven Multicast
  — Multicast message is being sent to each Finger
  — Range contained in the message tells a node to which other nodes it has to forward the Multicast message to
Chord – Protocol Based QoS

> Peers are ordered by QoS classes.
  — Clockwise monotonically decreasing QoS classes
  — There can be several links to lower QoS classes
  — Core based tree with root having the minimal ID
Chord Optimizations - Why?

> Chord in its original form performs pretty bad especially if the interval between joining nodes is small
  — Successor pointers are very inaccurate
  — Fingers are updated too slow: routing is very inefficient
  — The joining node doesn't get its predecessor by itself it has to wait for its predecessor to stabilize until it gets to know it
  — Unusable for Multicasting
    - Many nodes don't get the Multicast messages
    - Many hops
  — Difficult to maintain QoS: the node with the smallest ID should be root

> Original Chord performs better if stabilize is called more often
  — Higher traffic requirements due to more synchronisation
  — Many problems remain unsolved
Improvement Visualization

> No Improvement
  — Nodes join in 0.2s interval
  — Nodes stay for at least 25 seconds
  — Random leave interval: 10 seconds
  — Stabilization all 2 seconds

> Improved chord
  — Nodes join in a 0.2s interval
  — Nodes stay for at least 5 seconds
  — Random leave interval: 10 seconds
  — Stabilization all 5 seconds but if it were turned off the result would be the same
Chord Optimizations - Fingers

> Problem: Fingers are updated too slow if a node just joined

> Solution

— Many fingers will point to the first successor
  1. Skip all fingers pointing to the successor
  2. If a new successor is encountered, refresh fingers according to 1
  3. Refresh the next position not pointing to the successor
  4. Start stabilization timer

— Get Finger table from new successors
— Further improvements considering RTT optimization will be handled later
Chord Performance - Number of Hops
Chord Performance – RTT to Root
Chord Performance – % Received Msgs
Chord Performance – Fan Out

![Graph showing Fan Out vs Number of Nodes]

- **Average**: The blue line represents the average fan out across different node counts.
- **Median**: The green line represents the median fan out.
- **Min**: The orange line represents the minimum fan out.
- **Max**: The red line represents the maximum fan out.

The graph illustrates how fan out behaves as the number of nodes increases.
Chord Performance – QoS fulfilled
End to End Delay

- It would be nice to have the end-to-end delay as a QoS parameter
- Problems
  - Delays vary with time
    - Reordering of the Multicast tree is needed
    - The reordering may lead to RTT problems of other nodes
  - Not all Overlays care about the underlying network
  - Sender driven Multicast: the receiver doesn't have any control from where it receives the Multicast messages
  - Receiver driven Multicast: the sender doesn't have control over its fanout
End-to-End Delay Sender Driven Multicast

> Chord - Optimize Fingers
  — FixFinger: Add a finger in the vicinity of the desired chordID
  — No RTT guarantees because Multicast messages don't come from the fingers
  — Overall RTTs optimized

> CAN - Landmarks
  — Joining Nodes could be placed near the landmark with the lowest RTT
  — As with Chord, the overall RTTs will be minimized and therefore the RTT to root as well
  — No RTT guarantees
End-to-End Delay Receiver Driven Multicast

- QoS ping (ping to Multicast root)
- Chord
  - Receiver driven fingers: nodes look up a parent supporting the required RTT to root
  - Drawbacks:
    1. Root is optimal parent for every node -> high fanout
    2. Unicast routing slower: nodes at the end of the circle will have less fingers!
  - Solutions
    1. Look for a parent around my chordID/2
    2. Maintain a second “fingerTable” for the Multicast distribution only so that unicast routing remains untouched
End-to-End Delay Receiver Driven Multicast

> CAN – split driven Multicasting
  - New nodes join at a coordinate where their RTT to root is fulfilled
  - Splitting node will be parent of the new node
  - No change of parents after splitting
  - Ask children if they can add the new joined node as parent
End-to-End-Delay Receiver Driven Multicast

> NICE
  — The RTTs are already optimized
  - Joining node pings each cluster leader and then joins the cluster having the smallest RTT
  - Cluster leader change: If the RTT to root is too high it will look for a cluster leader, which guarantees its RTT requirements

> Scribe/Pastry
  — Generate routing table with QoS pings
  — Root = node with the id closest to the group id
  — Tree built by joining the multicast group (message routed to the root)
  - Route joining message to a node supporting my RTT to root requirements
End-to-End Delay Framework

> No RTT optimizations
> Links are established between the roots of the layers

1. Improvement
> Links between roots
> Hop optimization
  — Less hops often indicate a lower RTT to root
  — No guarantees
  — Fan out problem
End to End delay framework

2. Improvement

- **Uplink**
  - Incoming gate
  - Is the root of the QoS class

- **Downlink**
  - Outgoing gate
  - Can be changed by a lower QoS class

> **Advantages**
- Root of a QoS class has more possibilities to choose a parent
- Fan out can be reduced
- RTT's may be optimized

> **Disadvantages**
- Hops increased
- The framework has no knowledge about the underlying P2P network
  - Every root has to provide a neighbour table
  - Down and Uplink have to know each other in the framework as well
Outlook

> Chord delay optimization
  — Optimize Finger delays
  — Implement receiver driven Multicast
> Implement dynamic behaviour
> Test Chord using the framework, help adapting the framework if it doesn't work
> Provide end-to-end delay guarantee support in the framework
> Evaluation: Protocol dependent / independent QoS
> Write the thesis
Questions