Prof. Dr. Thomas Schmidt
HAW Hamburg, Dept. Informatik
Raum 480a, Tel.: 42875 - 8452
Email: t.schmidt@haw-hamburg.de
Web: http://inet.haw-hamburg.de/teaching

Technik & Technologie vernetzter Systeme

Teil 2: P2P-Kommunikation: Chord mit Broadcast (3. & 4. Praktikum)

Projekt: Implementierung eines verteilten Spiels "Schiffe Versenken" (ohne Churn).

Die Spielteilnehmer bilden gemeinsam eine Chord DHT. Nach Abschluss der Chord Initialisierung unterteilt jeder Knoten/Spieler seinen Bereich im DHT-Adressraum in I äquidistante Intervalle (numerisch überzählige Adressen werden dem numerisch letzten Intervall zugeschlagen) und platziert $S < I$ Schiffe darin. Aufgabe des Spiels ist es, die Schiffe der Mitspieler zu "versenken" - wer zuerst das letzte Schiff eines Mitspielers getroffen hat, hat gewonnen.

Im Einzelnen ist der Spielablauf wie folgt:

1. Der Spieler mit dem größten Knotenschlüssel beginnt (Besitzer des Schlüssels 2^{160} -1). Er sendet ein `retrieve` auf einen Schlüssel seiner Wahl.
2. Der Empfänger des `retrieve` prüft, ob sich ein Schiff in demselben Intervall befindet, wie der verlangte Schlüssel. Ist dies der Fall, ist das Schiff zerstört, anderenfalls ging der "Schuss" ins Leere.
3. Der Empfänger benachrichtigt alle Teilnehmer (per Broadcast) über den Zerstörungsversuch (angerouteter Schlüssel) und seinen Ausgang. Dabei kann ein Schiff nicht zweimal zerstört werden, d.h. jeder weitere Schuss auf das selbe Feld geht ins Leere.
4. Hiernach ist der Empfänger an der Reihe und darf ein `retrieve` auf einen Schlüssel seiner Wahl initiieren.
5. Derjenige Knoten/Spieler, der als Erster richtigerweise bemerkt, dass er das letzte Schiff eines Mitspielers zerstört hat, hat gewonnen.

Das Spiel findet vor dem Hintergrund unvollständiger (Chord Routing Tabellen), aber wachsender (Broadcasts der Ergebnisse) Informationen statt. Es obliegt jedem einzelnen Spieler, diese Informationen zu sammeln und so geschickt auszuwerten, dass er eine erfolgreiche "Ratestrategie" anwenden kann.

Die Kernkomponenten des "Verteilten Schiffe Versenkens" bestehen aus:

- ➔ Der Chord Middleware (vorhanden).
- ➔ Einem Broadcast-Nachrichtentyp und einem Broadcast-Routing (s.u.).
- ➔ Einer Auswertung der bisher gelernten, verteilten Spielverhältnisse.
- ➔ Einer Strategie, die gelernten Spielverhältnisse auszunutzen.

Implementierungsplan:

Eine vorbereitete Implementierung (modifiziertes OpenChord) finden Sie als Zusatzmaterial auf der Webseite. OpenChord ist mithilfe von Java RMI implementiert, welches eine mögliche Realisierung darstellt (typische Alternativen sind TCP Sockets oder Message Passing). Bitte unterscheiden Sie jedoch sorgfältig zwischen der Chord-Schicht und der darunterliegenden Kommunikationsschicht (RMI).

Workpackage 1. Rekursiver Broadcast on Chord

Ein rekursiver Broadcast on Chord soll - analog zu dem Prefix-Flooding on Pastry - erreichen, dass Knoten, die ein Broadcast-Paket empfangen, dieses so lange und so gezielt weiterleiten, dass schließlich alle Teilnehmer das Broadcast-Paket genau einmal erhalten haben. Hierzu muss der Flooding-Algorithmus an die Intervallstruktur von Chord wie folgt angepasst werden:

> Jeder Knoten leitet ein Broadcast-Paket mit RangeHash **X** an alle ihm bekannten Knoten (mit aktualisiertem Range) zwischen seiner **ID** und **X** weiter.

Im Detail:

1. Der Startknoten sendet das Broadcast-Paket an alle (unterschiedlichen) Knoten seiner Finger Table, wobei der den jeweils darauffolgenden Finger-Table Eintrag in das RangeHash - Feld notiert.
2. Der Empfänger eines Broadcast-Pakets stellt dieses seiner Applikation zu und leitet es an alle Knoten weiter, welche gemäß seiner Finger Table zwischen seiner eigenen ID und dem RangeHash liegen. Dabei trägt er analog den jeweils darauffolgenden Finger Table Eintrag in das RangeHash-Feld ein.
3. Liegen keine weiteren Knoten mehr zwischen der eigenen und der RangeHash ID, terminiert das rekursive Weiterleiten.

Hilfestellungen und Konventionen:

NotifyCallback

Damit die Spiele-Anwendung über eingehende Broadcast- und Retrieve-Aufrufe informiert wird und entsprechend reagieren kann, müssen Sie das NotifyCallback-Interface implementieren. Die Definition finden Sie im Package *<de.uniba.wiai.lspi.chord.com>*. Das Interface müssen Sie der Chord-Implementierung bekanntmachen bevor sie ein Chord-Netzwerk erstellen oder einem beitreten (siehe *ChordImpl.setCallback(...)*).

Broadcast Subsystem

Erweitern Sie das Chord-Interface sowie dessen Implementierung um einen Broadcast-Aufruf analog zu den vorhanden Aufrufen (insert, retrieve und remove).
Damit die Anwendung einen Broadcast-Inform ausführen kann, müssen die Methoden:

- `ChordImpl.broadcast(ID` *`target`*`, Boolean` *`hit`*`)`
- `NodeImpl.broadcast(Broadcast` *`info`*`)`

aus dem Package *<de.uniba.wiai.lspi.chord.service.impl>* implementiert werden. Die erste Methode ist Teil der API und wird von der Anwendung aufgerufen, sobald ein retrieve erfolgt ist. Anschließend muss für alle zutreffenden Knoten in der Finger-Table der Broadcast mittels RMI aufgerufen werden. Für die Implementierung des RMI-Broadcasts ist im Package *<de.uniba.wiai.lspi.chord.com>* die Klasse `Broadcast` definiert. Diese fasst die folgenden Informationen zusammen:

- `ID` *`range`*: der Weiterleitungsbereich für den rekursiven Broadcast (s.u.)
- `ID` *`source`*: der Knoten/Spieler auf den geschossen wurde
- `ID` *`target`*: Schlüssel bzw. Hash-ID auf den geschossen wurde
- `Integer` *`transaction`*: Transaktionsnummer, aufsteigend
- `Boolean` *`hit`*: gibt an, ob ein Schiff getroffen wurde (true) oder nicht (false).

Damit die Spiele-Anwendung auch über den Broadcast informiert wird, muss in der Methode *`NodeImpl.broadcast(...)`* auch der entsprechende Notify-Callback aufgerufen werden (s.o.). Weitere Hinweise:

- Hilfreiche Methoden für die Implementierung: `ChordImpl.getID()`, `ChordImpl.getPredecessorID()`, `ChordImpl.getFingerTable()`.
- Die Rückgabe-Liste von `getFingerTable()` ist nicht notwendiger Weise sortiert, allerdings implementiert die Klasse Node das Interface `Comparable`.
- Das Senden eines Broadcast muss parallel (asynchron) zum übrigen Programmablauf erfolgen.

Weitere Informationen

Die Originalaufrufe des Chord-Interface verwenden die Klasse *`Key`*, um Werte in der DHT abzulegen, der Key wird intern von Chord zu einer ID gehasht. Für das Spiel ist es erforderlich die Chord-ID selbst zu definieren und den entsprechenden Wert in der DHT abzufragen. Benutzen Sie daher die Methode *`retrieve(ID target)`* statt *`retrieve(Key key)`*.

***Meilenstein 1:* Broadcast ist funktionsfähig.**

Workpackage 2 - Informationsstrukturen und Strategie "Schiffe Versenken"

Entwerfen und implementieren Sie eine geeignete Daten- und Informationsstruktur, welche die im Spielverlauf erlauschten Informationen auswertet und zwischenspeichert. Errichten Sie auf dieser Informationsstruktur eine Strategie, welche Ihre (erfolgreichen) Spielzüge ermittelt.

Verwenden Sie eine *einfach aufsteigende TransaktionsID*, um den Gewinner eindeutig identifizieren zu können. D.h. der Startknoten/spieler beginnt mit einer zufällig gewählten TransaktionsID, welche für jede *neue* Nachricht erhöht wird.

***Meilenstein 2:* Sie können an dem verteilten Spiel teilnehmen.**

Workpackage 3 – Status-Anzeige am IoT-Knoten

Zur Visualisierung des eigenen Spielerstatus soll die RGB-LED eines Sensorknotens verwendet und mittels des IoT-Protokolls CoAP angesteuert werden. Nach jedem Beschuss wird parallel zum Broadcast der aktuelle Spielerstatus, d.h. die Anzahl der versenkten Schiffe als Farbkodierung an den Sensor-Knoten geschickt. Verwenden Sie dafür den aus Labor1 bekannten Aufbau für Szenario 1 und senden Sie vom Arbeitsplatz-Rechner (wo das Spiel läuft) einen CoAP PUT-Request mit entsprechendem Inhalt an den Sensor-Knoten.

Verwenden Sie die RGB-LED des Sensor-Knotens zur Anzeige des Spielstatus wie folgt:

- Grün: Spieler bereit, alle Schiffe unversehrt
- Blau: mindestens eines, aber weniger als 50 % der Schiffe wurden versenkt
- Violett: mehr als 50%, aber nicht alle der Schiffe wurden versenkt
- Rot: alle Schiffe wurden versenkt

Verwenden Sie für die Implementierung das Eclipse Californium Framework, siehe: http://www.eclipse.org/californium/. Hilfreiche Code-Beispiele finden Sie im zugehörigen Github-Repository, unter: https://github.com/eclipse/californium.

***Meilenstein 3:* IoT-Anbindung funktionsfähig**

***Workpackage 4* - Test und Optimierung für *I = 100 und S = 10*.**

Testen Sie Ihre Lösungen im Verbund und optimieren Sie Ihre Spielstrategie, so dass Ihr Programm die besseren Spielzüge durchführt.

***Meilenstein 4:* Sie gewinnen zuverlässig.**

Wettkampftag: Mittwoch, 11. Januar 16:00 - 18:30 Uhr (Vorbereitung vorab!)

Die Wettkampfgewinner (1. bis 3. Platz) werden von der Firma Ferchau Engineering mit zusammen 1.000 € Preisgeld unterstützt

Abgabe: (per Email wie bei Aufgabe 1)

1. Kommentierter Programmcode
2. Kurzdokumentation Ihres Vorgehens bei der Spielstrategie

Deadline: 05. Januar