Chord Protocol - Stabilization

Antonio Bucchiarone

Fondazione Bruno Kessler, Trento – Italy
bucchiarone@fbk.eu

19 November 2018

Case Study: Stabilization

- Chord requires that a node should periodically perform a stabilization operation.
  - Stabilization ensures that the node's successor is still the next closest node on the ring.
  - If a closer node is found, the the successor pointer needs to be updated (state change).

ChordProtocol

```
public class ChordProtocol implements EDProtocol, Comparable<ChordProtocol> {

    public Node node;

    public BigInteger predecessor;
    public BigInteger[] fingerTable, successorList;
    public BigInteger chordId;

    public int fingerToFix=0;

    public ChordProtocol(String prefix) {
    }


    public void processEvent(Node node, int pid, Object event) {

    private void receive(ChordMessage msg){

    private void send(ChordMessage msg, BigInteger destID){


    public void onRoute(ChordMessage msg){

    public void onFinal(ChordMessage msg){

    public void onSuccessorFound(ChordMessage msg){

    public void onNotify(ChordMessage msg){

    public void join(Node myNode) {

    public void findSuccessor(BigInteger nodeToAsk, BigInteger id, String label){


    private BigInteger closestPrecedingNode(BigInteger id) {

    private ArrayList<BigInteger> getFullTable(){

    public void notify(BigInteger nodeId){

    public void stabilize() {

    public void fixFingers(){
```

- PeerSim Library

```
import peersim.core.CommonState;
import peersim.core.Network;
import peersim.core.Node;
import peersim.edsim.EDProtocol;
import peersim.transport.Transport;
```

Chord Message

```
public class ChordMessage {

    public static final int LOOK_UP=0;
    public static final int FINAL=1;
    public static final int SUCCESSOR=2;
    public static final int SUCCESSOR_FOUND=3;
    public static final int NOTIFY=4;

    private String label;
    private int type;
    private BigInteger sender;
    private Object content;
    private ArrayList<String> path = new ArrayList<String>();

    public ChordMessage(int type, Object content) {
        this.type = type;
        this.content = content;
    }


    public int getType(){..

    public BigInteger getSender() {..

    public void setSender(BigInteger sender){..

    public Object getContent() {..


    public void addToPath(BigInteger chordId){..
    public ArrayList<String> getPath(){..


    public String getLabel() {..


    public void setLabel(String label) {..

    public boolean isType(int type){..


}
```

receive Message

```
public void processEvent(Node node, int pid, Object event) {

    ChordMessage msg = (ChordMessage) event;
    receive(msg);
}

private void receive(ChordMessage msg){
    switch(msg.getType()){
    case ChordMessage.LOOK_UP:
        onRoute(msg);
        break;
    case ChordMessage.SUCCESSOR:
        onRoute(msg);
        break;
    case ChordMessage.SUCCESSOR_FOUND:
        onSuccessorFound(msg);
        break;
    case ChordMessage.FINAL:
        onFinal(msg);
        break;
    case ChordMessage.NOTIFY:
        onNotify(msg);
        break;

    }

}
```

receive Message

```
public void processEvent(Node node, int pid, Object event) {

    ChordMessage msg = (ChordMessage) event;
    receive(msg);
}

private void receive(ChordMessage msg){
    switch(msg.getType()){
    case ChordMessage.LOOK_UP:
        onRoute(msg);
        break;
    case ChordMessage.SUCCESSOR:
        onRoute(msg);
        break;
    case ChordMessage.SUCCESSOR_FOUND:
        onSuccessorFound(msg);
        break;
    case ChordMessage.FINAL:
        onFinal(msg);
        break;
    case ChordMessage.NOTIFY:
        onNotify(msg);
        break;

    }

}
```

Chord Initialization

```java
public class ChordInitializer implements NodeInitializer, Control {

    private static final String PAR_PROT = "protocol";
    private static final String PAR_TRANS = "transport";

    int pid = 0;
    int tid = 0;


    public ChordInitializer(String prefix) {
        pid = Configuration.getPid(prefix + "." + PAR_PROT);
        tid = Configuration.getPid(prefix + "." + PAR_TRANS);
        Utils.initialize(pid, tid);
    }
```

```java
public static void initialize(int pid, int tid){
    if(!initialized){
        PID = pid;
        TID = tid;
        SUCC_SIZE = Configuration.getInt("SUCC_SIZE", 4);
        M = Configuration.getInt("M", 10);
        receivedMessages = new ArrayList<ChordMessage>();
    }
}
```

```java
public boolean execute() {
    ArrayList<BigInteger> ids = Utils.generateIDs(Network.size());

    for (int i = 0; i < Network.size(); i++) {
        Node node = (Node) Network.get(i);
        ChordProtocol cp = Utils.getChordFromNode(node);
        cp.node = node;
        cp.chordId = ids.get(i);
        Utils.NODES.put(cp.chordId, cp);
        cp.fingerTable = new BigInteger[Utils.M];
        cp.successorList = new BigInteger[Utils.SUCC_SIZE];
    }
    NodeComparator nc = new NodeComparator(pid);
    Network.sort(nc);
    myCreateFingerTable();
   // printNeighs();
    return false;
}
```

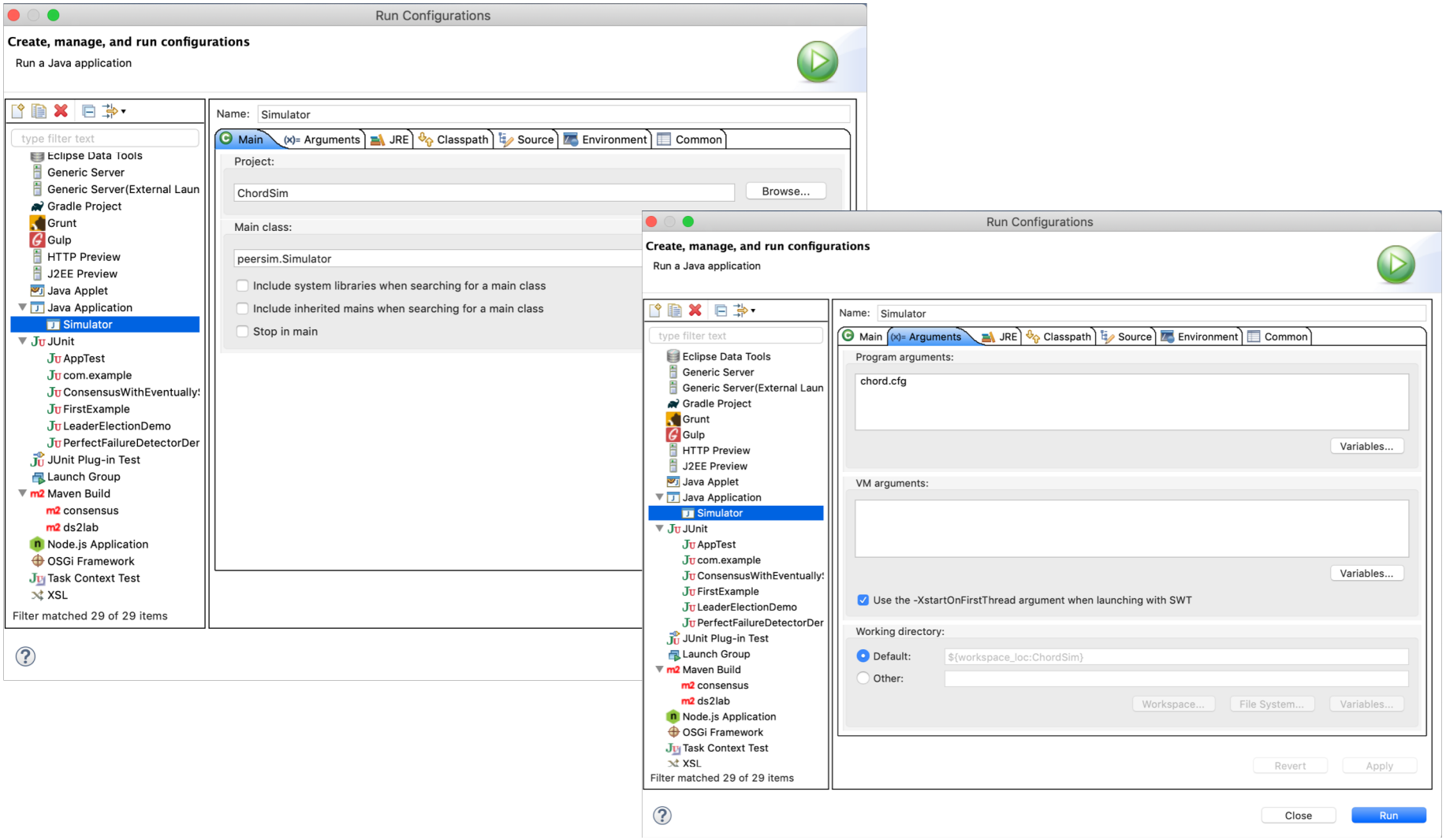
Stabilization

```
public void stabilize() {
    for(BigInteger succ: successorList){
        if(succ != null && Utils.isUp(succ)){
            successorList[0] = succ;
            findSuccessor(succ, succ, "successor stabilize");
            return;
        }
    }
    System.err.println("All successors of node " + this.chordId + " are down!");
    System.exit(1); //something went totally wrong

}
```

```
public void findSuccessor(BigInteger nodeToAsk, BigInteger id, String label){
    ChordMessage predmsg = new ChordMessage(ChordMessage.SUCCESSOR, id);
    predmsg.setLabel(label);
    predmsg.setSender(chordId);
    send(predmsg, nodeToAsk);
}
```

Eclipse – Run Configuration with Arguments



PeerSim Configuration

```
# PEERSIM CHORD

CYCLES 1000 #10^6
SIZE 20 #100
M 10 #size fingers table
SUCC_SIZE 4

random.seed 12345678
simulation.endtime CYCLES
simulation.logtime 10^6

simulation.experiments 1

network.size SIZE

protocol.tr UniformRandomTransport
{
    mindelay 0
    maxdelay 0
}

protocol.my ChordProtocol
{
    transport tr
}

control.traffic TrafficGenerator
{
    protocol my
    step 1
    #step 100
    #from 100
    #until 201
    #step 100
}


init.create ChordInitializer
{
    protocol my
    transport tr
}

control.maintain ChordMaintainer
{
    step 10500
}



control.observer ResultObserver
{
    protocol my
    #step 90000
    at CYCLES
    FINAL
}
```