24 November 2017

Homeworks Distributed Systems

TomP2P a P2P framework/library

Carmine Spagnuolo, Ph.D.

Homeworks Prerequisites

- Concurrent object-oriented programming fundamental → Threads mechanisms and Observer pattern are required.
- Distributed systems fundamental → Distributed Hash Tables (DHT) is required.
- Programming language → Java version 7 or greater.
- Project management → Apache Maven.
- Programming IDE → Eclipse is optional.
- Versioning control Git → GitHub a Web-based Git version control repository hosting service.

What is TomP2P?

TomP2P is a P2P framework/library

- Implements DHT (structured), broadcasts ([un]structured), direct messages (can implement super-peers).
- NAT handling: UPNP, NATPMP, new addition: relays, hole punching (work in progress).
- Direct/indirect (tracker/mesh) storage.
- Direct/indirect replication (churn prediction and rsync).
- Modes: key,value / multi-key (versioned) value.

What is TomP2P?

TomP2P extends DHT

- Distributed hash table concept ← put(key,value) / get(key).
- Extended DHT operations:
  - put(key1,key2,value);
  - put prepare / put confirm;
  - add(value);
  - digest(key) / bloomfilters / versions;
  - get(key) + bloomfilters.

Source

`https://tomp2p.net/doc/M05-1up_v2.pdf`

Introduction to Put and Get operations

Create a project and setup the `pom.xml` if you are using MAVEN:

```
    <repositories>
        <repository>
            <id>tomp2p.net</id>
            <url>http://tomp2p.net/dev/mvn/</url>
            </repository>
    </repositories>
    <dependencies>
        <dependency>
            <groupId>net.tomp2p</groupId>
            <artifactId>tomp2p-all</artifactId>
            <version>5.0-Beta8</version>
        </dependency>
    </dependencies>
```

Introduction to Put and Get operations

Example application that maps a name to a value.

```
1       public class ExampleSimple {
        final private PeerDHT peer;
3       public ExampleSimple(int peerId) throws Exception {
            peer=new PeerBuilderDHT(new PeerBuilder(Number160.createHash(peerId)).
5                       ports(4000 + peerId).start()).start();
            FutureBootstrap fb=
7               this.peer.peer().bootstrap().inetAddress(InetAddress.
                                getByName("127.0.0.1")).ports(4001).start();
9           fb.awaitUninterruptibly();
            if(fb.isSuccess()) {
11              peer.peer().discover().peerAddress(fb.bootstrapTo().
                        iterator().next()).start().awaitUninterruptibly();
13          }
        }
15  ...
```

Introduction to Put and Get operations

Example application that maps a name to a value.

```
1       private String get(String name) throws ClassNotFoundException,
            IOException {
3           FutureGet futureGet = peer.get(Number160.createHash(name)).start();
            futureGet.awaitUninterruptibly();
5           if (futureGet.isSuccess()) {
                return futureGet.dataMap().values().iterator().
7                   next().object().toString();
            }
9           return "not found";
        }
11      private void store(String name, String ip) throws IOException {
            peer.put(Number160.createHash(name)).
13              data(new Data(ip)).start().awaitUninterruptibly();
        }
15  }
```

Introduction to Put and Get operations

Example application that maps a name to a value.

```
1
    public static void main(String[] args) throws NumberFormatException,
3       Exception {
        ExampleSimple dns = new ExampleSimple(Integer.parseInt(args[0]));
5       if (args.length == 3) {
            dns.store(args[1], args[2]);
7       }
        if (args.length == 2) {
9           System.out.println("Name:" + args[1] + " IP:" + dns.get(args[1]));
        }
11  }
}
```

Introduction to Put and Get operations

- Run this example, you first have to start the well known peer on port 4001:
  `java ExampleSimple 1 test.me 192.168.1.1`
- Then you can add as many other clients as you want:
  `java ExampleSimple 2 test.me`
- The output should look something like:
  `Name:test.me IP:192.168.1.1`

Peer building

Create the peer with a KeyPair.

```
Random rnd = new Random();
Peer peer = new PeerBuilder(new Number160(rnd)).ports(4001).start();
```

Create the peer with a public / private key.

```
KeyPairGenerator gen = KeyPairGenerator.getInstance("DSA");
KeyPair pair1 = gen.generateKeyPair();
Peer peer = new PeerBuilder(pair1).ports(4001).start();
```

More peers. Create a new peer and listen to another port, or attach a new peer to an existing port.

```
Peer another = new PeerBuilder(new Number160(rnd)).masterPeer(peer).
                    ports(4002).start();
```

Peer building

Discover if our peer is behind a NAT and if it is, TomP2P needs to configure NAT via UPNP.

```
FutureDiscover future = another.discover().
                            peerAddress(peer.peerAddress()).start();
future.awaitUninterruptibly();
```

Non-blocking communication

Send objects on P2P.

```
1       PeerDHT pdht = new PeerBuilderDHT(another).start();
        Data data = new Data("test");
3       Number160 nr = new Number160(rnd);
        FuturePut futurePut = pdht.put(nr).data(data).start();
5       futurePut.awaitUninterruptibly();
```

TomP2P uses non-blocking communication, a future object is used to keep track of future results. Thus, a `get().start()`, `put().start()`, or `add().start()` returns immediately and the future object is used to get the results from those operations.

Non-blocking communication

- There are two options to get the data from the future object. The first is by blocking and waiting for the result to arrive, which can be either `await()` or `awaitUninterruptibly()`.
- The second option is to add a listener, which gets called whenever a result is ready.

```
futureGet.addListener(new BaseFutureAdapter<FutureGet>() {
    public void operationComplete(FutureGet future) throws Exception {
        if(future.isSuccess()) {
            System.out.println("success");
        } else {
            System.out.println("failure");
        }
    }
});
```

Direct messages

TomP2P can send direct messages in an RPC-style to other peers. There are two types of send() functions defined in TomP2P: sending objects and sending raw data.

```
peer.peer().objectDataReply(new ObjectDataReply() {
    public Object reply(PeerAddress sender, Object request)
    throws Exception {
            System.err.println("I'm " + peer.peerID() +
                               " and I just got the message
                               [" + request+ "] from " +
                               sender.peerId());
    return "world";
 }
});
FutureDirect futureDirect = _dht.peer().sendDirect(peer).
                               object(_obj).start();
futureDirect.awaitUninterruptibly();
```