ECE 454/750: Distributed Computing
Tutorial 2: Advanced Thrift

TA: Hua Fan

h27fan@uwaterloo.ca

May 19, 2015 05:30-06:20, QNC 2502

Outline

In this tutorial we will discuss:

- Slides and source code available: https://ece.uwaterloo.ca/~h27fan/ece454_750/tut2.tar.gz
- Tutorial 1: review
- Tutorial 2: set up
- RPC review
- Generated code analysis
- Task 1: Multi-threading servers
- Task 2: Async Client
- Task 3: Multi-threaded client
- Task 4: Experiment small worker number

Tutorial 1: review

- tar xzvf thrift-0.9.2.tar.gz
- tar xzvf tut1.tar.gz
- cp -r tut1_src thrift-0.9.2/tutorial/
- cd thrift-0.9.2/lib/java && ant
- cd ../../tutorial/tut1_src/ && ant

Tutorial 2: set up

- tar xzvf thrift-0.9.2.tar.gz
- tar xzvf tut2.tar.gz
- cp -r tut2_src thrift-0.9.2/tutorial/
- cd thrift-0.9.2/lib/java && ant
- cd ../../tutorial/tut1_src/ && ant
- cp log4j.properties build

Set up(cont.)

- Another RPC used today:
  - // delay by sleep in seconds, before return.
  - i32 DelayAdd(1:i32 num1, 2:i32 num2, 3:i32 delay_s)

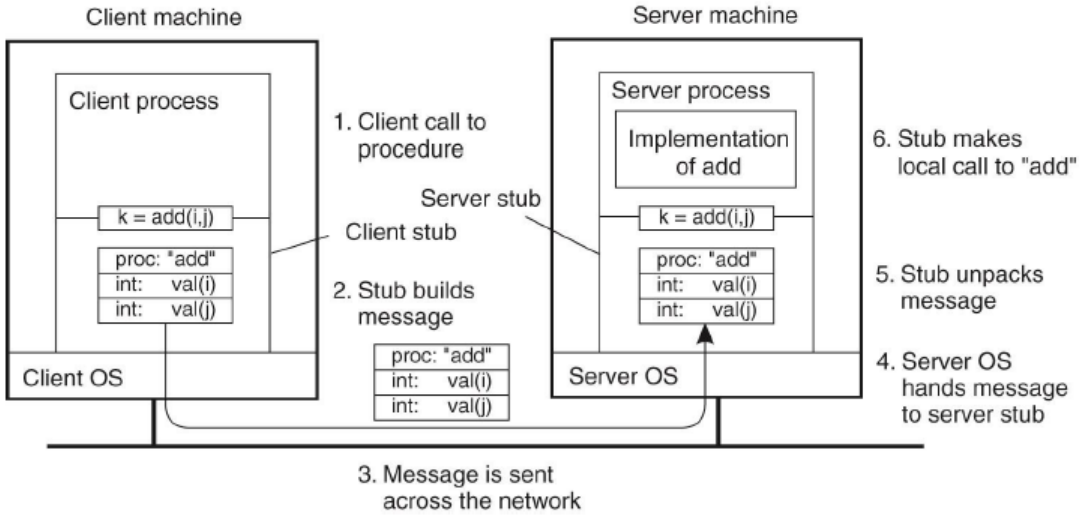
```
public int DelayAdd(int n1, int n2, int delay) {
  System.out.println("Sleep " + delay + " seconds.");
  try{Thread.sleep(delay * 1000);}catch(InterruptedException e){System.out.println(e);}

  System.out.println("Delayadd(" + n1 + "," + n2 + ")");
  return n1 + n2;
}
```

RPC review(lecture slide)

Steps 1-6 illustrated

Note: the act of packing parameter values into a message in step 2 is called **parameter marshalling**.



Analysis gen-java/tutorial/Myservice.java

- Client class takes the role of client stub

```
public static class Client extends org.apache.thrift.TServiceClient implements Iface {

public int add(int num1, int num2) throws org.apache.thrift.TException
{
  send_add(num1, num2);
  return recv_add();
}
```

- Server takes the role of Server stub, which get message then call processor

TSimpleServer

```
client = serverTransport_.accept();
if (client != null) {
  processor = processorFactory_.getProcessor(client);
  inputTransport = inputTransportFactory_.getTransport(client);
  outputTransport = outputTransportFactory_.getTransport(client);
  inputProtocol = inputProtocolFactory_.getProtocol(inputTransport);
  outputProtocol = outputProtocolFactory_.getProtocol(outputTransport);
  if (eventHandler_ != null) {
    connectionContext = eventHandler_.createContext(inputProtocol, outputProtocol);
  }
  while (true) {
    if (eventHandler_ != null) {
      eventHandler_.processContext(connectionContext, inputTransport, outputTransport);
    }
    if(!processor.process(inputProtocol, outputProtocol)) {
      break;
    }
  }
```

Analysis (cont.)

- Processor will call handler

```
public add_result getResult(I iface, add_args args) throws org.apache.thrift.TException {
  add_result result = new add_result();
  result.success = iface.add(args.num1, args.num2);
  result.setSuccessIsSet(true);
  return result;
}
```

- Handler Implements your procedures!

MyserviceHandler.java

```
public class MyserviceHandler implements Myservice.Iface {
```

Myservice.java

```
public interface Iface {

  public int add(int num1, int num2) throws org.apache.thrift.TException;

  public Item getItem(int key) throws org.apache.thrift.TException;

  public void putItem(Item item) throws org.apache.thrift.TException;

}
```

Naming rules

- Service name: as defined in .thrift file.
- Interface name: Service_name.Iface or .AsyncIface

```
public interface AsyncIface {

  public void add(int num1, int num2, org.apache.thrift.async.AsyncMethodCallback resultHandler) t
```

- Synchronous client name: Service_name.Client
- Asynchrounous client name : Service_name.AsyncClient
- Processor: Service_name.Processor

Outline

In this tutorial we will discuss:

- Slides and source code available: https://ece.uwaterloo.ca/~h27fan/ece454_750/tut2.tar.gz
- Tutorial 1: review
- Tutorial 2: set up
- RPC review
- Generated code analysis
- Task 1: Multi-threading servers
- Task 2: Async Client
- Task 3: Multi-threaded client
- Task 4: experiment small worker number

Review thrift servers(from lecture slide)

Multi-threading: server

- Server threading depends on the implementation.
- **TSimpleServer** uses a single thread and blocking I/O.
- **TNonblockingServer** uses a single thread and non-blocking I/O. It can handle parallel connections but executes requests serially just like TSimpleServer.
- **THsHaServer** uses one thread for network I/O and a pool of worker threads. It can process multiple requests in parallel.
- **TThreadedSelectorServer** uses a pool of threads for network I/O and a pool of worker threads for request processing.
- **TThreadPoolServer** uses one thread to accept connections and then handles each connection using a dedicated thread drawn from a pool of worker threads.

https://github.com/m1ch1/mapkeeper/wiki/Thrift-Java-Servers-Compared

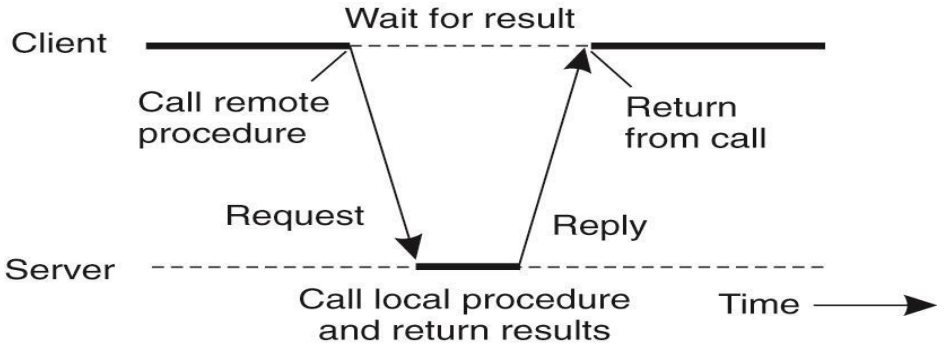
Task 1: Multi-threading servers THsHaServer

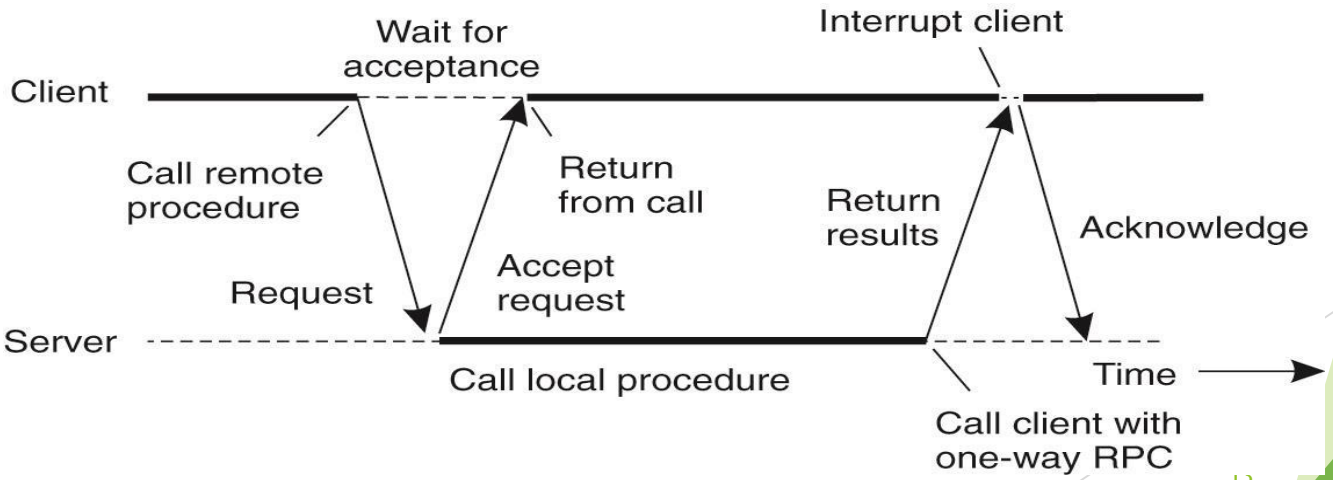
```
handler = new MyserviceHandler();
processor = new Myservice.Processor(handler);

TNonblockingServerSocket socket = new TNonblockingServerSocket(9090);
THsHaServer.Args arg = new THsHaServer.Args(socket);
arg.protocolFactory(new TBinaryProtocol.Factory());
arg.transportFactory(new TFramedTransport.Factory());
arg.processorFactory(new TProcessorFactory(processor));
arg.workerThreads(5);

TServer server = new THsHaServer(arg);
server.serve();
System.out.println("HsHa server started.");
```

Review async client(from lecture slides)





Task 2: Async Client

- Callback object: implement onComplete, onError
- Callback template type: procedure FUNCTION => FUNCTION_call

```
static class AddCallBack
  implements AsyncMethodCallback<Myservice.AsyncClient.DelayAdd_call>

  public void onComplete(Myservice.AsyncClient.DelayAdd_call add_call)
      try {
          long result = add_call.getResult();
          System.out.println("Add from server: " + result);
      } catch (TException e) {
          e.printStackTrace();
      }
      finish = true;
  }

  public void onError(Exception e) {
      System.out.println("Error : ");
      e.printStackTrace();
      finish = true;
  }
}
```

Async Client (cont.)

```
TProtocolFactory protocolFactory = new TBinaryProtocol.Factory();
TAsyncClientManager clientManager = new TAsyncClientManager();
TNonblockingTransport transport = new TNonblockingSocket("localhost", 9090);

Myservice.AsyncClient client = new Myservice.AsyncClient(
    protocolFactory, clientManager, transport);

client.DelayAdd(200, 700, 5, new AddCallBack());
System.out.println("After Send Async call.");

int i = 0;
while (!finish) {
    try{Thread.sleep(1000);}catch(InterruptedException e){System.out.println
    i++;
    System.out.println("Sleep " + i + " Seconds.");
}
```

Task 3: Multi-threaded client

static CountDownLatch latch = new CountDownLatch(5);

```
for(int i = 0; i < 5; ++i){
    System.out.println("Send request i = " + i);
    new Thread() {
        public void run() {
            try {
                TProtocolFactory protocolFactory = new TBinaryProtocol.Factory();
                TAsyncClientManager clientManager = new TAsyncClientManager();
                TNonblockingTransport transport = new TNonblockingSocket("localhost", 9090);
                Myservice.AsyncClient client = new Myservice.AsyncClient(
                    protocolFactory, clientManager, transport);
                client.DelayAdd(100, 200, 3, new AddCallBack(latch, transport));
            } catch (TException x) {
                    x.printStackTrace();
            } catch (IOException e) {
                    e.printStackTrace();
            }
        }
    }.start();
    System.out.println("After Send request i = " + i);
}
boolean wait = latch.await(30, TimeUnit.SECONDS);
```

Multi-threaded client(cont.)

```
static class AddCallBack
  implements AsyncMethodCallback<Myservice.AsyncClient.DelayAdd_call> {

  private CountDownLatch latch;
  private TNonblockingTransport transport;

  public AddCallBack(CountDownLatch latch, TNonblockingTransport transp) {
      this.latch = latch;
      this.transport = transp;
  }
  public void onComplete(Myservice.AsyncClient.DelayAdd_call add_call) {
      try {
          long result = add_call.getResult();
          System.out.println("Add from server: " + result);
      } catch (TException e) {
          e.printStackTrace();
      } finally {
          transport.close();
          latch.countDown();
      }
  }

  public void onError(Exception e) {
      System.out.println("Error : ");
      e.printStackTrace();
      latch.countDown();
  }
}
```

Task 4: experiment small worker number

- Set worker thread = 2, while 5 client threads concurrent calling
- Demo