

Second Assignment – Tutorial lecture

INF5040 (Open Distributed Systems)

Tien Dat Le (dattl@ifi.uio.no)



Department of Informatics
University of Oslo

October 16, 2017

Group Communication System

- ▶ Services provided by group communication systems:
 - Abstraction of a Group
 - Multicast of messages to a Group
 - Membership of a Group
 - Reliable messages to a Group
 - Ordering of messages sent to a Group
 - Failure detection of members of the Group
 - A strong semantic model of how messages are handled when changes to the Group membership occur



Spread

- ▶ An open source toolkit that provides a high performance messaging service that is resilient to faults across local and wide area networks
- ▶ Does not support very large groups, but does provide a strong model of reliability and ordering
- ▶ Integrates a membership notification service into the stream of messages
- ▶ Supports multiple link protocols and multiple client interfaces
- ▶ The client interfaces provided with Spread include native interfaces for Java and C
 - Also non native Pearl, Python and Ruby interfaces



Spread

- ▶ Provides different types of messaging services to applications
 - Messages to entire groups of recipients
 - Membership information about who is currently alive and reachable
- ▶ Provides both ordering and reliability guarantees



Level of Service

- ▶ When an application sends a Spread message, it chooses a **level of service for that message**.
- ▶ The level of service selected controls what kind of ordering and reliability are provided to that message.

Spread Service Type	Ordering	Reliability
UNRELIABLE MESS	None	Unreliable
RELIABLE MESS	None	Reliable
FIFO MESS	Fifo by Sender	Reliable
CAUSAL MESS	Causal (Lamport)	Reliable
AGREED MESS	Total Order (Consistent w/Causal)	Reliable
SAFE MESS	Total Order	Safe



Ordering

- ▶ **None**
 - No ordering guarantee
 - Any other message sent with “None” ordering can arrive before or after this message
- ▶ **FIFO by sender**
 - All messages sent by this connection of FIFO ordering are delivered in FIFO order
- ▶ **Causal (Lamport) – See TvS 6.2**
 - All messages sent by all connections are delivered in an order consistent with “Causal” order (Lamport)
 - Consistent with FIFO ordering
- ▶ **Total Order (Consistent w/Causal)**
 - All messages sent by all connections are delivered in the exact same order to all recipients
 - Consistent with Causal order



Reliability

▶ Unreliable

- The message may be dropped or lost
- The message will not be recovered by Spread

▶ Reliable

- The message will be reliably delivered to all recipients who are members of the group to which the message was sent
- Spread will recover message to overcome any network losses

▶ Safe

- The message will only be delivered to a recipient if the daemon that recipient is connected has the message
 - The daemon should know all other Spread daemons in the network
- If a membership change occurs, and as a result the daemon cannot determine whether all daemons in the old membership have the message, then the daemon will deliver the Safe message after a TRANSITIONAL_MEMBERSHIP message.



Download and Installation

- ▶ Spread can be downloaded from
 - <http://www.spread.org/>
 - <http://www.cnds.jhu.edu/>.
- ▶ I recommend you to download both the binaries and source
 - Binaries contains the daemon distribution for different architecture
 - Source contains the Spread source code and example apps
 - Example of configuration file (*sample.spread.conf*) in the docs folder.
- ▶ Run the Spread daemon
 - You can use the Spread monitor to check the status of your daemon : `./spread -l y -c spread.conf` and `./spmonitor`



Java Interface to Spread Toolkit

- ▶ The Spread library consists of a package, “spread”
 - 10 classes.
 - Eclipse: Import the classes into a new Java project or use the Ant build file
- ▶ Main classes:
 - *SpreadConnection*, which represents a connection to a daemon,
 - *SpreadGroup* which represents a spread group
 - *SpreadMessage*, which represents a message that is either being sent or being received with spread.



Connecting/Disconnecting

- ▶ To establish a connection to a spread daemon

```
SpreadConnection connection = new SpreadConnection();  
connection.connect(InetAddress.getByName(  
    "daemon.address.com"),  
    1010, "privatename", false, false);
```

Server Address

Port

Connection unique name

- ▶ To terminate the connection to the daemon,

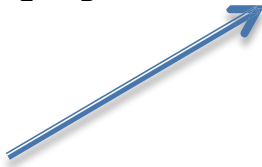
```
connection.disconnect();
```



Joining/Leaving

- ▶ To join a group on the connection

```
SpreadGroupgroup = new SpreadGroup();  
group.join(connection, "group");
```



Spread connection on
which the group is joined



Name of the group to join

- ▶ To leave a group,


```
group.leave();
```

Multicasting

- ▶ To multicast a message to one or more groups

```
SpreadMessage message = new SpreadMessage();
```

```
message.setData(data);  
message.addGroup("group");  
message.setReliable();
```



This creates a new outgoing message

- ▶ To send the message,

```
connection.multicast(message);
```

Receiving

► To receive a message

```
SpreadMessage message = connection.receive();
```

receive() will block until a message is available

```
if(message.isRegular())  
    System.out.println("New message from " +  
        message.getSender());  
else  
    System.out.println("New membership message from " +  
        message.getMembershipInfo().getGroup());
```

return a *MembershipInfo* object, which provides information about the membership change



Message Factory

- ▶ A utility included with the java interface to spread
- ▶ An object of the MessageFactory class is used to generate any number of outgoing messages based on a default message.

```
messageFactory = new MessageFactory(message) ;
```

- ▶ To change the default at a later time

```
messageFactory.setDefault(message) ;
```

- ▶ To get a message from the message factory

```
SpreadMessage message = messageFactory.createMessage() ;
```



Listeners

- ▶ An alternative way of receiving messages
- ▶ Interfaces:
 - BasicMessageListener
 - AdvancedMessageListener

```
connection.add(listener);
```

- ▶ After being added to a connection, the listener will be alerted whenever a new message is received on the connection.
- ▶ To remove a listener from the connection

```
connection.remove(listener);
```



Exceptions

- ▶ When an error occurs in a Spread method, a `SpreadException` is thrown
 - Eg. `receive()` is called on a `SpreadConnection()` object before `connect()` is called on that object
- ▶ Any method that is declared as throwing a `SpreadException` must be placed within a try-catch block

```
try{  
    connection.multicast(message);  
}catch (SpreadException e)  
{   e.printStackTrace()  
    ();   System.exit(1);  
}
```



Second Programming Assignment

- ▶ A distributed application that models a replicated bank account
- ▶ Implementation
 - “Replicated state machine” paradigm
 - Group communication
- ▶ System Architecture
 - A standard Spread server (daemon)
 - A set of clients representing the replicas



Tasks

- ▶ Create a connection to a Spread server.
- ▶ Initialize the balance on the account to 0.0.
- ▶ Join a group whose name is *<account name>*.
- ▶ Wait until it detects that *<number of replicas>* clients have joined the group.
 - Application starts from this point, but should handle the dynamic addition of new replicas.
- ▶ The client should receive and analyze messages about membership changes.



Commands to be accepted by the client

- ▶ **balance**
 - Print the current balance on the account.
- ▶ **deposit or withdraw <amount>**
 - Increase or decrease the balance by <amount>.
 - On all the replicas in the group.
- ▶ **addinterest <percent>**
 - Increase the balance by <percent> percent of the current value.
 - On all the replicas in the group.
- ▶ **sleep <duration>**
 - Let the client to do nothing for <duration> seconds. It is only useful in a batch file.

exit



Testing environment

- ▶ Spread is installed in all linux ifi computers
 - /local i.e. /local/bin, /local/lib etc.
- ▶ Daemon configuration/addresses for the assignment
 - You can use any ifi machine to start your own server:
 - `spread -l y -c spread.conf`

```
Spread_Segment 127.0.0.255:4803
{ localhost <machine
  address>
}
```

- Example address:
 - `rubin.ifi.uio.no 4333`
- You can run the server and the clients in the same machine but it is a good idea to test your application in a distributed environment with more than 3 distributed clients.

Synopsis of running the client

- ▶ For java

- `java accountReplica <server address> <account name> <number of replicas> [file name]`

- ▶ You can also develop a graphical user interface if you do not want to use the command line.

