SWIM through the NATs

Jim Dowling, Salman Niazi

March 31, 2015

The goal of this project is to implement and evaluate a decentralized membership service that provides information about *all* the the participating processes. A membership service can easily be implemented using a heatbeat mechanism, where each process periodically sends a ping message to all the processes in the system. Failures are detected when a process does not responds to the ping message. However, this solution does not scale well when the number of processes in the system grow because heatbeat based mechanisms requires O(n^2) messages in each round.

The project is split into two phases. In the first phase of the project you will implement SWIM: *S*calable *W*eakly-consistent *I*nfection-style Group *M*embership *P*rotocol [1] using Kompics that scales to at least hundreds of peers in simulation. In phase two you will modify the system to make with work on the Open Internet.

1 Phase 1: Implementing and Evaluating SWIM

Warm Up: Ping Pong

The bootstrap mechanism provides a small subset of existing peers in the system. A new peer injects information about itself by randomly pinging one of the peers provided by the bootstrap server. Implement a basic ping pong mechanism. Select a random peer from the list of alive nodes and send a ping message. When a peer receives a ping message it responds back with a pong message. Using the log messages make sure that the ping mechanism is working.

Task 1: Membership discovery

In this task you will implement the piggybacking mechanism to disseminate information about new peers in the system. In the pong messages the peers piggyback information about the changes (new/dead/suspected peers) in the system. Upon receiving the pong message the new information is merged with the local state stored in one or more local lists.

The authors recommend exchanging the new information $\lambda.logn$ times, where $\lambda \geq 1$. In the SWIM paper the authors provide (p.6) some information about how to maintain different lists of peers. However, you are free to implement your own data structures for maintaining different states of the peers. Using the log messages make sure the system works and converges in logarithmic time.

At this stage we do not recommend limiting the size of the information exchanged in each message. For now you can piggyback all state changes in one message.

Task 2: Basic SWIM failure detector (FD)

Now, modify the Kompics experiments to introduce failures in the system by stopping/disconnecting random peers in the systems. When a peer does not respond to ping messages it is declared dead. Note, at this stage there is no need to implement the *suspect* sub protocol. Using the log messages make sure that the information about the dead peers are discriminated to all the peers and the system converges.

Task 3: SWIM FD Contd.

Due to temporary network link outages and process unresponsiveness a process may fail to respond to a ping request. Peers that fail to respond to ping messages are not declared dead right away. Such peers are declared to be *suspected*. The information about the suspected peers is gossiped around. Suspected peers are treated just like alive peers until they are confirmed to have failed. If the state of the suspected process is not contested then the peer is declared dead after some predefined timeout. A suspected state of the process is overridden when 1) The peer finds out that other peers have started to suspect it, in such a case in can inject *Alive* messages in the system 2) When a peer successfully manages to ping a suspected peer, it declares that the peer is no longer suspected.

Implement the suspect mechanism and make sure that it is working accordingly i.e. first the suspect information is gossiped around and then the suspected peers are declared dead after the predefined timeout.

Task 4: Message Ordering

In the task 2 failures were introduced by disconnecting a peer component. You can resume normal execution of the component by reconnecting the peer to the network. Modify the Kompics experiment to randomly disconnect some peers. Reconnect some processes and observe if the application converges.

As the messages are randomly ordered, in some experiment runs you will notice many false positive failure detections. Implement the incarnation counter mechanism as described in the paper to handle this problem.

Task 5: K-Indirect Pings

Implement K-Indirect ping sub protocol as described in the SWIM paper.

Task 6: Analyze the system

Run the multiple experiments with varying amount of failures and network sizes (10, 20, 50, 100, 200 ...) and report how long the system takes to converge.

Task 7: Limiting the information exchanged

So far we have not limited the amount of information piggybacked over the ping pong messages. In large systems it may not be practical to piggyback all information over one message. Now you will repeat the previous task by varying amount of information piggybacked over single message. Set the information exchange limit to $(1, 2, 3...log.n, n.log.n)$ and report how long the system takes to converge.

2 Phase 2: SWIM Through the NATs

SWIM assumes that there is direct connection between any two peers in the system. Some studies have shown that up to 80% of the peers in the peer-to-peer overlays on the Internet are behind NATs and firewalls. Network Address Translation (NAT) gateways enable many machines to share a small number of public IP addresses, but at the cost of preventing external hosts from directly connecting to hosts behind the NAT.

In the rest of the document open peers refers to peers that have public IP and they are accessible by any peer in the system. A natted peer is behind a NAT device and it is not directly visible to the peers on the Internet. Your next task is to run SWIM on the Open Internet where some peers are behind NAT devices. Two popular mechanisms to overcome this problem are Hole Punching and Message Relaying. These mechanisms will be discussed in detail in the following lectures. Due to time limitations you will not implement hole punching mechanisms and rely solely on message relaying to communicate with the natted peers.

Up till now all the peers in the system were open i.e. all the peers could communicate with each other. In the remaining tasks you will start the system with both open and natted peers. You have been provided a simple $NatTraverser$ layer that hides the complexities of the communicating with natted peers. When a natted peer is started it selects a random open peer as its relay server. When any node wants to send a message to a natted peer the $NatTraverser$ layer forwards the message to the relay server for the natted peer, and the relay server forwards the message to the natted peer.

The problem with the current implementation is that the when an open peer dies then all the natted peers that were using the open peer as a relay are disconnected. In this task you will fix this problem by adding the support for *multiple* relay servers. You will implement the following

- Connect and Start $Croupier$ component. Croupier [2] is a peer sampling service that provides random samples of open and natted peers in they system.
- The $NatTraverser$ layer periodically send heart beats to the relay servers. When it detects that a relay server is dead then it selects a new open peer provided by Croupier as the relay server. The information about the new relay server needs to be disseminated to all the peers in the system.
- The $NatTraverser$ layer notifies the SWIM layer about the changes in the list of relay servers. The SWIM component injects the new updated $address$ information in the system.

3 System Diagram

Figure 1 shows a system diagram of the scaffolding code. Some of the components are partially implemented , while Croupier is functional. A simple experiment, SwimScenario.java, is provided. Feel free to write your own new events and experiments.

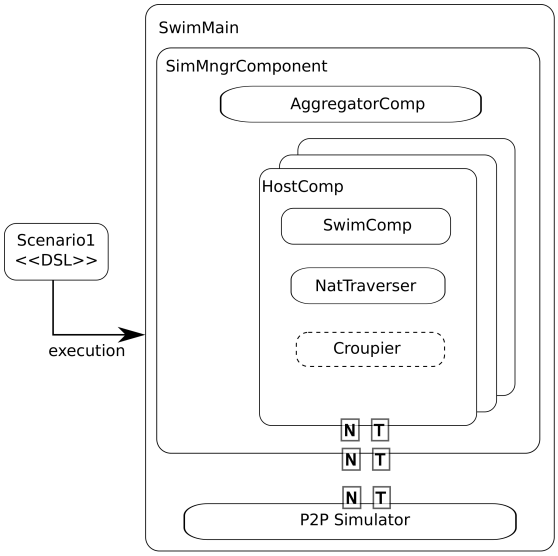

Figure 1: System Diagram

4 Kompics, Maven

The sample source code provided is written in Kompics/Java. If you are not familiar with Kompics, please do the tutorials available at

- http://www.ict.kth.se/courses/ID2203/programming_assignments.html
- http://kompics.sics.se
- https://github.com/kompics/kompics

The build system used for source-code provided to you is maven. Maven has built-in support in Netbeans, and there's a m2eclipse plugin for Eclipse. Quick tutorial is here http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

5 Source Code

There is some scaffolding code available to enable you to implement the protocol without too much effort. The Source Code is available on github. You don't have to, but I recommend you create your own account on github and fork the project. This way, if you find any bugs, you can fix them and send a pull request to me. Bonus points if you do that! The homepage on github is here: https://github.com/smkniazi/id2210-vt15

To get started without forking the project, you can clone it using the following command:

> $ git clone https://github.com/smkniazi/id2210-vt15.git

6 Project Deliverables

Submit a project report and the source code of your implementations for grading.

6.1 Report

We recommend that you use Latex (http://www.maths.tcd.ie/ dwilkins/LaTeXPrimer/) to produce your report. Your report should adhere to the following guidelines:

- Submit your report in a PDF file.
- Your report should not contain more than 5 A4 pages.
- In the analysis of your plots, identify the plot with its label. For example, write "from figure 2, it can be seen that ..." instead of "from my figure it can be seen that ...".

Your report should contain following sections

- Report results for task 6 and 7.
- When the system is complete repeat the experiments with different network sizes with varying ratio of open and natted peers.

6.2 Source code

The requirements for the submitted source code are the following:

- The whole project should be submitted as a single compressed file to Bilda. Use either zip or tar+gzip.
- The code should be compilable. Uncompilable source code will not be graded.
- Your source code should be well-readable.
- Please name your variables appropriately and add comments to your code when needed.
- Make sure that the reported simulation results are reproducible.
- Include all experiment scenario files for the reported experiments.

References

[1] Abhinandan Das, An Das, Indranil Gupta, and Ashish Motivala. Swim: Scalable weakly-consistent infection-style process group membership protocol. In *In Proc. 2002 Intnl. Conf. Dependable Systems and Networks (DSN '02*, pages 303–312, 2002.

[2] Jim Dowling and Amir H. Payberah. Shuffling with a croupier: Nat-aware peer-sampling. In *Proceedings of the 2012 IEEE 32Nd International Conference on Distributed Computing Systems*, ICDCS '12, pages 102–111, Washington, DC, USA, 2012. IEEE Computer Society.