CS4103-DS Practical 2: Coding Project

120008200

April 22, 2016

1 Introduction

An implementation of the Bully algorithm has been developed in Java. A runnable Jar file has been built in the project root which is used by a set of test scripts to run experiments.

2 Running Instructions

The program arguments are specified as follows:

java -jar bully.jar uuid port nodeType timeout config-file [Initiator]

nodeType is one of the following:

- Normal - a node without any failures.
- Timing - a node with sporadic delay sending messages.
- Omission - a node which fails to send some messages.
- Crash - a node which crashes.

The final parameter **Initiator** indicates that a node should start the election process.

To launch a set of nodes, bash scripts have been provided with various different configurations of nodes and failures. To test a normal run, execute the **normal_test.sh** script in the root directory. These scrips launch the processes and collate the logs from all of the nodes into a single file. For example, **normal_test.sh** would put the results into a file named **normal_results.txt**.

3 Design

3.1 Architectural Overview

The bully algorithm implementation has been split into two main packages: bully and bully.fault. The bully package contains all of the code for the bully algorithm itself, and the bully.fault package contains a set of helper classes which facilitate the addition of the different node failures (Crash, Omission, Timing).

The main class is Bully.java. The Bully class parses the command line arguments, then reads the configuration file to determine which other nodes are running. Each node is added to a dictionary, mapping uuids to node objects.

Once the setup has been completed, if the current node is the initiator, it starts an election, otherwise it starts listening on a socket for incoming messages.

3.2 Faulty Nodes

The bully.fault package contains helper classes for each of the node failure types. The crash class is run in a thread alongside the main program, which kills the program when **Math.random()** exceeds a certain threshold. The Omission class provides a static function **Omit()** which will return true if the node is to omit the next message. The timing class provides a static function **artificialDelay()** which delays the current thread (and therefore the message sending).

4 Testing

4.1 Normal Execution

The script **normal_test.sh** runs six nodes (defined in config1.txt) where node 3 is the initiator, and none of the nodes have faults. The results for this run are piped into the file named **normal_results.txt**.

Note: The example results directory in the project root contains a set of results from past runs of all of the tests.

4.2 Omission

To test the omission failure, the **omission_test.sh** script has been created. This test case is identical to the normal test case but with the introduction of an Omission faulty node (Node 5). This node will sometimes choose not to send messages which simulates the loss of messages in transmission. Analysis of the logs will identify which messages are omitted. The line:

[2016-04-22 11:55:14:202 — 5] Internal: Omission.

shows that an omission was made (taken from **example_results/omission_results.txt**).

4.3 Timing

The **timing_test.sh** script has been created to test timing related faults. This test case is identical to the normal test case but with the introduction of a Timing faulty node (Node 5). This introduces artificial delay in transmission of messages. The line:

[2016-04-22 12:04:42:505 — 5] Internal: Artificial delay: 1070

shows that an artificial delay was introduced of 1070 milliseconds (taken from **example_results/timing_results.txt**).

4.4 Crash

The **crash_test.sh** script has been created to test faults where a node crashes. This test case is identical to the normal test case but with the introduction of a crashing node (Node 5). Node 5 is very likely to crash during the election process. The line:

[2016-04-22 12:14:36:485 — 5] Internal: Process crashed

shows that process 5 crashed (taken from **example_results/crash_results.txt**).

4.5 Results

All of the tests with faulty nodes ended up electing node 6 (the bully) as the leader, with the exception of the timing test. The delay in the timing test caused the faulty node (node 5) to send RESULT to all of the nodes, after node 6 had already been elected. This could be mitigated by simply increasing the timeout (which was set to 100 ms).

These results show the Bully algorithm successfully electing a leader even with different types of faulty node introduced into the group.