Design Document

Submitted By: Gaurav Jain

Part A1: Details common to Coordinator and normal members of overlay

A node can start as a coordinator or as a normal member*(non coordinator). Each node (whether coordinator or normal member) has following threads:

1) messageHandler Thread:
This thread responds to a variety of messages like coordinator lookup messages, join messages, leave messages etc. by listening on a well-defined port (LISTEN_PORT in code). This thread in turn, spawns a new thread to handle a particular message. For example, when a node sends a 'JOIN' message to the coordinator, messageHandler thread at the coordinator spawns a new thread to handle this message which communicates with the joining member.

2) pingHandler Thread:
This thread performs different functions at different times depending on whether the node is currently acting as a coordinator or a normal member of the overlay. If the node is a coordinator, this thread is responsible for sending 'ping' message to all other members of the overlay. If the node is a normal member, then this thread responds with a 'pong' message to 'ping' message from the coordinator.

3) Main thread:
After spawning the above two threads, main thread goes into a loop collecting latency measurements (wrt other members in the overlay) at a regular interval of 30 s. Also, every 5 minutes, main thread sends the latency measurement to the coordinator (exception: the node itself is the coordinator OR coordinator is not known because election is in process). The main thread is also responsible for catching the Ctrl-C interrupt. On catching this interrupt, main thread breaks from the loop and sends a 'LEAV' (leave) message to the coordinator. Since, all child threads are created with the setdaemon = True attribute, all of them are killed (abruptly) as soon as the main thread exits. More elaborate cleanup of threads can be implemented, but setting threads as daemons has proved sufficient for our purpose.

Apart from the above threads, we also have a **SimpleHTTPServer** running on each of the nodes that provides easy access to files. Using SimpleHTTPServer, transfer of files (members.txt - Holding information on all members in the overlay) becomes trivial. Also, it allows having a look at logs (logs.txt) and latency measurements (latency.txt) from the comfort of a web-browser.

Part A2: Implementation details of Coordinator API

1) listMembers()
This is a utility function used by both coordinators as well as normal members to get a list of all the members currently in the overlay. The function simply reads from a ‘members.txt’ file maintained at each node.

2) addMember(nodeId)
This function appends the nodeId to the “members.txt” file maintained at the coordinator. Once updated, the “members.txt” file is then distributed to members in the overlay using distributeMembersFile(members) function. distributeMembersFile(members) sends a short ‘DWLD’ (download) message to the members. These members then download the updated “members.txt” file over HTTP (SimpleHTTPServer serving files in home directory). Note that addMember(nodeId) function is called in response to ‘JOIN’ message from a member, and as such is executed only at the coordinator.

3) removeMembers(nodeIdList)
The API here differs slightly from the specification. Unlike the provided specification, this function can take a list of members to be removed from the “members.txt” file at the coordinator. This function opens the members.txt file for writing and removes all members present in the list provided as parameter. The updated file “members.txt” is then distributed using distributeMembersFile(members) function mentioned above, to members in the updated overlay.

This function apart from being called in response to ‘LEAV’ message sent by a leaving member, is also called by the coordinator itself, if a member doesn’t respond to ping messages. The coordinator adds such members to a list, and calls removeMembers() with this list as the parameter.

4) ping(nodeId, retry)
The ping function is implemented over UDP socket with a timeout of 10 s (Experiments with a timeout of 5 sec were found to raise false positives). Also, a retry value (=2) is specified. If the retries also timeout, nodeId is assumed to be disconnected and it is added to the list of members to be removed.

** Coordinator is maintained in a global variable at each node. Reads and writes on global variables are thread-safe operations (From Python Documentation)

Part A3: Implementation details of member API:

1) coordinatorLookup()

A member calls this function if it starts as a normal member. It contacts all the members listed in

seeds.txt file, sending them a ‘LKUP’ (lookup) message. If any of the nodes know the coordinator, they reply back with the coordinator name, which this node can adopt (set global variable Coordinator). If none of the nodes reply in first attempt, the member waits for COORDINATOR_PROBE_INTERVAL = 10 s, and sends ‘LKUP’ message to each of the seeds again. If a reply is still not forthcoming, the member declares itself as the coordinator and starts performing the duties of the coordinator (responding to JOIN, LEAV, LKUP messages etc.)

2) join(Coordinator)

A member calls this function if coordinatorLookup returns some other node (than itself). Member node sends a ‘JOIN’ message in response to which the coordinator calls addMember(nodeName).

3) leave()

This functionality is implemented inline in code in the form of sending a ‘LEAV’ message to the coordinator before exiting. Main thread is in the latency measurement gathering loop until it catches KeyboardInterrupt exception. The exception handler acknowledges the Ctrl-C press, and sets condition to break from the loop. The message is sent to the coordinator just before calling sys.exit().

PART B1: Coordinator failure detection

Corresponding to the ‘ping’ functionality of the coordinator, each member has a ‘pong’ functionality to reply back to ping messages from the coordinator. The pong(retry) is implemented over a UDP socket with timeout 10 s. Once the retries (=2) also timeout without having received any ‘ping’ from the coordinator, the member initiates election of a new coordinator by calling electCoordinator() function.

PART B2: Coordinator re-election

electCoordinator()
Each node has a list of active members in the overlay in members.txt file. All nodes are ranked lexicographically. Lexicographically smallest node(name) has the highest rank and gets preference over others in coordinator election. A node sorts the members in members.txt file and starts sending ‘LKUP’ (same as in coordinatorLookup) messages to all nodes that are higher ranked. As long as a higher ranked node replies (even with a NONE message - NONE corresponds to coordinator unknown), a lower ranked node never tries to become coordinator. However, the electCoordinator() at a member may be made to break in between, in case some other node decides to become the coordinator and sends a ‘NEWC’ (new coordinator) message

to all other members. In that case, every member abandons the search (electCoordinator()) to honour the ‘NEWC’ message from higher ranked node.

An interesting scenario is when a low-rank node starts as a coordinator and a high-rank node starts as a member. If the high-rank node somehow detects timeout of the low rank coordinator, it could very well elect itself as the new coordinator (if it is the highest rank active member) and announce to all other members. Any lower ranked node has no choice, but to respect the announcement of the higher ranked node, even if the lower ranked (original) coordinator never detected any timeouts.

PART B3: Latency sensor

Latency measurements are taken by the main thread at each node at regular interval of about 30 seconds. The round trip time is measured by using the system ping (os.popen) and parsing the results. For each latency measurement, an average of 5 pings is calculated. Every 5 minutes, each member sends latency measurements between itself and other nodes in the overlay, to the coordinator. Also, the latency measurements are logged in a file “latency.txt”

Appendix:

Following are the most important messages exchanged among the members of the overlay:
1) JOIN:
Sent by a member to the coordinator to join the overlay

2) LEAV:
Leave message sent by a departing member of the overlay to the coordinator.

3) LKUP:
Coordinator lookup message sent during bootstrap coordinatorLookup and during electing new coordinator (electCoordinator)

4) DWLD:
Download message sent by the coordinator to trigger members to download the updated copy of members.txt file.

5) NEWC:
New Coordinator announcement sent by a newly elected coordinator to other members in the overlay

6) LTNC:
Latency message (containing latency measurements) sent by a normal member of the overlay to the coordinator.