ECE454/750: Distributed Computing

Assignment 1 Design Report

Blaise Juvenal Leskowsky
20434269

Lai Kit So
20432346

RPC Implementation:

This implementation of this project used synchronous remote procedure calls. This type of RPC was implemented in the early stages of development for two reasons: first, development of this type of RPC was fairly simple, and second, reference information and documentation for using synchronous RPCs were more readily available. Additionally, since the gossip callback was not very light-weight, using asynchronous calls would be detrimental to performance.

Multi-threading Servers:

For multi-threading on the servers, the servers used the ThreadedSelectorServer. This was essentially a Half-Sync/Half-Async with a synchronous worker thread pool to process requests and a selector thread pool which handled non-blocking I/O. Since this system was making a large number of connections to different nodes, performance should benefit from using a server with non-blocking I/O.

Protocols:

The protocol used for communication was the TBinaryProtocol, which encoded numeric values as binary, as opposed to converting to plaintext. This protocol was chosen for its simplicity. The TCompactProtocol was considered, but it was determined that since this implementation that relies on sending a fairly large number of Events through the system, the extra overhead of compacting messages would be too high and lead to a degradation in performance.

For the purposes of nodes communicating their knowledge of other nodes in the system, our gossip protocol implemented a one-way RPC, gossip(Event e). Each node would broadcast an Event (either Arrival or Dead event type)containing the new Node's IP address and the source IP address combination of the sender to each of their known nodes whenever a change in the node's list was detected (i.e. a node attempted to connect to a seed). Each node who received this event would forward the Node Arrival Event to its own known nodes, changing the source IP address of the message (similar to an Internet Protocol datagram header). This forwarding of Events would terminate at a node when the node receiving the Event already has the IP address and the source IP address in its own node list. Essentially, the algorithm would allow the nodes eventually converge to a stable state where all the nodes in the system would be known to every other node.

Load Balancing:

To achieve loading balancing, each FE would select a BE using a weighted random algorithm, using the BE's number of cores as the weight. This achieved load balancing since BEs were assigned work using a uniform distribution, with nodes having more cores having the higher probability of receiving the request.

Crash Failure Detection/Handling:

To handle crash failure and detection, whenever an exception was caught while issuing a request to a node, the node who detected the error would attempt to inform the failure to at least one of the seed nodes. This utilized the gossip algorithm again, except a"Dead Node" Event would be broadcasted. On receiving this event, nodes would proceed to remove the node from their known list, forwarding the Event again in the same way as a Node Arrival case.