

Distributed Key-Value Storage Algorithm

CS181E — Distributed Systems

Assignment 5

Alejandro Frias

Ravi Kumar

David Scott

March 30, 2014

Algorithm Description

Table 1: Caption

Message Description	Erlang Pattern
Request to backup a store. Sent and received by <code>store_handler</code> . The receiver backs up the data, then notifies the OW of the store's success, and the old value. Sent by a <code>store_handler</code> when it receives a <code>store</code> message from one of its storage processes.	{Pid, Ref, backup_store, Key, Value, ProcessID}
Messages about keys. Sent and received by <code>store_handler</code> . If <code>Ref</code> is in the list of the receiver's in-progress computations, the computation is over, and send the result <code>ComputationSoFar</code> back to the OW. Otherwise, perform a step of the computation and forward the message to the next node's <code>store_handler</code> .	{Pid, Ref, *_key, ComputationSoFar}
Leave request. Sent by OW; received by <code>store_handler</code> or <code>storage_process</code> . If received by a <code>storage_process</code> , just forward it to the <code>store_handler</code> . If received by a <code>store_handler</code> , just kill the entire node.	blah
Joining behind. Received and sent by <code>store_handler</code> . When received, send all stored backup data to <code>Pid</code> , then delete all backup data for processes numbered less than <code>NodeID</code> . Is sent by a new node with ID <code>NodeID</code> to the pre-existing node that is sequentially after it.	{Pid, joining_behind, NodeID}

This table continues on the next page...

Table 1: Caption

Message Description	Erlang Pattern
Joining in front. Received and sent by <code>store_handler</code> . When receiving such a message: if your ID is equal to <code>DestID</code> , kill the data storage processes that the new node is now running (i.e. the ones numbered from <code>NodeID</code> to the ID of the node after the new one.	<code>{joining_front, NodeID, DestID}</code>
Node with <code>NodeID</code> died. Received by <code>store_handler</code> , sent by a <code>gen_server</code> listener started by that particular <code>store_handler</code> . When such a message is received, the <code>store_handler</code> changes the node's ID to <code>NodeID</code> , then uses all of the backup data it's holding to start up new data storage processes. Then it deletes the backup data, and sends a <code>backup_request</code> message around the ring.	<code>blah</code>
Backup node data. Received and sent by <code>store_handler</code> . When received, add all the data to existing backup data. Sent to a node <i>A</i> 's successor when <i>A</i> 's predecessor died and <i>A</i> is taking over for its predecessor.	<code>{backup_node, Data}</code>
Backup request. Received and sent by <code>store_handler</code> . If it is received on the node with <code>DestID</code> , send each of this node's <code>storage_processes</code> an <code>all_data</code> message. After compiling all of the results from those requests, send all of this node's stored data to this node's successor node in a <code>backup_node</code> message. If this node is not <code>DestID</code> , just forward the request message. Initially sent by a node which stepped into the void left by a node that died.	<code>{backup_request, DestID}</code>
All data request message. Received by <code>storage_process</code> and sent by <code>store_handler</code> . When received by a <code>storage_process</code> , respond with an <code>all_data_send</code> message containing all this <code>storage_process</code> 's data.	<code>{all_data, Pid}</code>
All data send message. Received by <code>store_handler</code> and sent by <code>storage_process</code> . The <code>store_handler</code> adds the received data to a <code>backup_request</code> message that's in progress; if it's the last response that was being waited for, send the <code>backup_request</code> .	<code>{all_data, Data, Pid}</code>

Correctness