Distributed Key-Value Storage Algorithm
CS181E — Distributed Systems
Assignment 5

Alejandro Frias Ravi Kumar David Scott

April 4, 2014

Algorithm Description

Table 1: Caption

Message Description	Erlang Pattern
Request to backup a store. Sent and received by `store_handler`. The receiver backs up the data, then notifies the OW of the store's success, and the old value. Sent by a `store_handler` when it receives a `store` message from one of its storage processes.	{Pid, Ref, backup_store, Key, Value, ProcessID}
Messages about keys. Sent and received by `store_handler`. If `Ref` is in the list of the receiver's in-progress computations, the computation is over, and send the result `ComputationSoFar` back to the OW. Otherwise, perform a step of the computation and forward the message to the next node's `store_handler`.	{Pid, Ref, *_key, ComputationSoFar}
Leave request. Sent by OW; received by `store_handler` or `storage_process`. If received by a `storage_process`, just forward it to the `store_handler`. If received by a `store_handler`, just kill the entire node.	blah
Joining behind. Received and sent by `store_handler`. When received, send all stored backup data to `Pid`, then delete all backup data for processes numbered less than `NodeID`. Is sent by a new node with ID `NodeID` 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 `store_handler`. When receiving such a message: if your ID is equal to `DestID`, kill the data storage processes that the new node is now running (i.e. the ones numbered from `NodeID` to the ID of the node after the new one.	{joining_front, NodeID, DestID}
Node with `NodeID` died. Received by `store_handler`, sent by a `gen_server` listener started by that particular `store_handler`. When such a message is received, the `store_handler` changes the node's ID to `NodeID`, 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 `backup_request` message around the ring.	blah
Backup node data. Received and sent by `store_handler`. When received, add all the data to existing backup data. Sent to a node A's successor when A's predecessor died and A is taking over for its predecessor.	{backup_node, Data}
Backup request. Received and sent by `store_handler`. If it is received on the node with `DestID`, send each of this node's `storage_process`es an `all_data` 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 `backup_node` message. If this node is not `DestID`, just forward the request message. Initially sent by a node which stepped into the void left by a node that died.	{backup_request, DestID}
All data request message. Received by `storage_process` and sent by `store_handler`. When received by a `storage_process`, respond with an `all_data_send` message containing all this `storage_process`'s data.	{all_data, Pid}
All data send message. Received by `store_handler` and sent by `storage_process`. The `store_handler` adds the received data to a `backup_request` message that's in progress; if it's the last response that was being waited for, send the `backup_request`.	{all_data, Data, Pid}

Correctness