Distributed Ksdfsdf sdf sey-Value Storage Algorithm
CS181E — Distributed Systems
Assignment 5

Alejandro Frias Ravi Kumar David Scott

April 5, 2014

Algorithm Description

Table 1: Caption

Message Description	Erlang Pattern
Store Request. Sent by either OW or a `storage_process`to a `storage_process`to store `value` for `key`. These are only ever received by storage processes in the network. When an `storage_process`receives such a message, it first checks if the hashed key is equal to its own ID. If it is, then it stores the value for the specified key, and sends a `backup_store` message to its `store_handler`. If not, it forwards it to the closest storage process to the destination (closest here meaning nearest process *before* the destination, since processes can only send messages forward in the ring).	{Pid, Ref, store, Key, Value}
Stored Confirmation. Sent by an `store_handler`to OW after the corresponding request has been stored in the proper storage process and the data has been backed up in the `store_handler`sending the message.	{Ref, stored, Old_Value}

This table continues on the next page...

Table 1: Caption

Message Description	**Erlang Pattern**
Retrieve Request. Sent by OW and storage processes; received by storage processes. When a storage processes receives such a message, it checks if the hash of the key is equal to its own process ID. If it is, the storage process has the value for that key, and replies with a retrieve response. If not, it forwards it to the closest storage process to the destination (closest as defined above).	{Pid, Ref, retrieve, Key}
Retrieve Response. Sent by storage processes to the OW. After a storage process receives a retrieve request meant for it, it looks up the relevant value and reports it to the requesting process in the OW.	{Ref, retrieved, Value}
First Key Request. Sent by the OW to storage processes, and by storage processes to storage handlers. When a storage process receieves this, it will forward the message up to its storage handler. When such a message is received by a storage handler, it will start a first key computation by adding the ref to its list of ongoing computations and sending a First Key Computation message to the next node in the ring.	{Pid, Ref, first_key}
Last Key Request. Sent by the OW to storage processes, and by storage processes to storage handlers. When a storage process receieves this, it will forward the message up to its storage handler. When such a message is received by a storage handler, it will start a last key computation by adding the ref to its list of ongoing computations and sending a Last Key Computation message to the next node in the ring.	{Pid, Ref, last_key}
Num Keys Request. Sent by the OW to storage processes, and by storage processes to storage handlers. When a storage process receieves this, it will forward the message up to its storage handler. When such a message is received by a storage handler, it will start a num keys computation by adding the ref to its list of ongoing computations and sending a Num Keys Computation message to the next node in the ring.	{Pid, Ref, num_keys}

This table continues on the next page...

Table 1: Caption

Message Description	**Erlang Pattern**
Node List Request. Sent by the OW to storage processes, and by storage processes to storage handlers. When a `storage_process`receieves this, it will forward the message up to its `store_handler`When such a message is received by a `store_handler`, it will query the global registry for the list of nodes and report that data to the requester.	{Pid, Ref, node_list}
Request Result. Sent to the OW by a storage handler. This reports the result of a First Key, Last Key, Num Keys, or Node List Request to the original requester after the storage handlers have finished computing the result.	{Ref, result, Result}
Failure Notification. Sent to the OW by storage handlers or storage processes to notify the OW that a particular computation has failed.	{Ref, failure}
Leave Request. Sent by the OW to storage processes and by storage processes to storage handlers. When received by a `storage_process`, it forwards the message to its `store_handler`. When received by an `store_handler`, it immediately kills all storage processes on the node it is running on, and kills itself.	{Pid, Ref, leave}
Backup Store Request. Sent by `store_handler`and `storage_process`, and received by `store_handler`. If a `store_handler`receives this message from a `storage_process`, it forwards the message to the next `store_handler`. If an `store_handler`receives this message from another `store_handler`, it will back up the data in the message, then notify the OW of the store's success and the old value.	{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}

This table continues on the next page...

Table 1: Caption

Message Description	**Erlang Pattern**
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}
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.	{died, NodeID}
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_processes` 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}

This table continues on the next page...

Table 1: Caption

Message Description	**Erlang Pattern**
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