

Distributed Key-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 <code>storage_process</code> to a <code>storage_process</code> to store value for key. These are only ever received by storage processes in the network. When an <code>storage_process</code> 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 <code>backup_store</code> message to its <code>store_handler</code> . If not, it forwards it to the closest storage process to the destination (closest here meaning nearest process <i>before</i> the destination, since processes can only send messages forward in the ring). | {Pid, Ref, store, Key, Value} |
| Stored Confirmation. Sent by an <code>store_handler</code> to OW after the corresponding request has been stored in the proper storage process and the data has been backed up in the <code>store_handler</code> 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 receives 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 receives 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 receives 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 <code>storage_process</code> receives this, it will forward the message up to its <code>store_handler</code> . When such a message is received by a <code>store_handler</code> , 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 <code>storage_process</code> , it forwards the message to its <code>store_handler</code> . When received by an <code>store_handler</code> , it immediately kills all storage processes on the node it is running on, and kills itself. | {Pid, Ref, leave} |
| Backup Store Request. Sent by <code>store_handler</code> and <code>storage_process</code> , and received by <code>store_handler</code> . If a <code>store_handler</code> receives this message from a <code>storage_process</code> , it forwards the message to the next <code>store_handler</code> . If an <code>store_handler</code> receives this message from another <code>store_handler</code> , 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 <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} |

This table continues on the next page...

Table 1: Caption

| Message Description | Erlang Pattern |
|--|--|
| 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. | <code>{Pid, joining_behind, NodeID}</code> |
| 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>{died, NodeID}</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> |
| This table continues on the next page... | |

Table 1: Caption

| Message Description | Erlang Pattern |
|---|------------------------------------|
| 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