Distributed Systems Assignment # 5

Authors: James Reinke and Will Tachau

- The Algorithm in Summary
    - The algorithm we have implemented for saving this Peer to Peer network takes as simplistic of a model as possible. Each node in our environment is represented by one non-storage process. Each of these non-storage processes is responsible for storing the dictionaries of the node above them in sequential order. This ensures that when a node goes down, those nodes are migrated to the new node that already holds their data. This allows us to efficiently transfer the data back to the migrated storage processes. When a node goes down our monitor warns us and we send a backup to be stored on the preceding node. Messages sent from this node can reach the preceding node in O(log(2^m)) time using our chord algorithm. We implemented in the chord algorithm in our function recipient. It finds the biggest jump a process can make without passing the target process while still obeying the rule of message passing neighbors. Storage processes are responsible for their own dictionary and no one else's. They are also responsible for sending a store message to non-storage processes on the correct node. For snapshots, we send a message around in a ring starting with the process that received the message that requires a snapshot (first key, last key). The message runs around the circle, moving from each process node one number at a time. They pass along the message and send their dictionary information item by item to the process that started the snapshot algorithm.
- States
    - Non-Storage Processes
        - Remain in a single listening state. Unless terminated, they resume listening after performing a finite number of actions
    - Storage Processes
        - Has two listening states. One for listening to the outside controller and other processes. Another is a listener for the snapshot algorithm, where the process determines if it has collected the full snapshot or not.
- Information Stored by Each Process
    - Non-Storage processes store a backup of every single processes' information located on the next node (ascending order by ID). They can backup process data when deleted processes are migrated to the hosting node. This allows for good locality of information.
- Message Types
    - Store
        - Sent by the controller or another process to store a value in a given storage process. When the storage process receives the

message, it updates its list and sends a message with the old value to the PID of the controller that it has received and updated its memory. If it is not intended for the storage process, it passes it along.

- Backup
  - Sent by a storage process after storing a new element into its dictionary. It sends it to the storage process who's ID matches the number of the node it wants its information stored on, namely the node preceding it. When the correct storage process receives it, it then sends the information to the storage process (the node) for storage. If a process receives this message that is it not intended for, it passes it along using our efficient chord message-passing algorithm.
- Retrieve
  - Received by a storage process from the controller to send it a value. It uses the hash function to find the storage process who has the key and value and forwards the message. When that process gets the message, it sends the result to the controller.
- First key
  - A storage process who receives this message enters a snapshot state, where it asks every other process to send the collector their first key in lexicographic order. It then takes the first key from that collection
- Last key
  - Performed much like first key. The process sends a message around in a circle for each process to send the collector their last key in lexicographic order. It then chooses the last key among this set.
- Num_keys
  - The receiving storage process asks every other process to send it the number of keys they currently have and then adds those numbers up.
- Node_list
  - The receiving process does a lookup on our table to find all of the nodes in our system in a list by ID.
- Snapshot
  - Sent by the snapshot collector to every other node to signal they it wants a snapshot of the contents in its dictionary.
- Imdying
  - Sent by a node that was told to leave that reaches its way to the node that will consume its processes (the node below it in order by ID).
- Dump
  - Sent by a known dying process to a non-storage process to store the contents of its dictionary.
- Sendbackup

        - Received by a non-storage process to replace it's backup dictionary with this new set
    - Makeproc
        - Tells the node (our only non-storage process) to generate a new process with process number and dictionary
    - Erasebackup
        - Sent by a storage process to a non-storage process to erase the contents of its dictionary
    - Dict_item
        - Sent by a process to the collector containing an item from their dictionary
    - Complete
        - Sent by a storage process when the contents of its dictionary have been completely flushed to the collector process.

- General Principles of the Algorithm
    - Maintains local manage of data
        - nodes that die send their processes to a neighboring node that already contains a backup of the processes' data
    - Keeps messages small
        - entire dictionaries are never sent, which could overflow the erlang messaging size
    - Fast message passing
        - Based on the communication rules of the chord protocol, we send messages in the fast manner possibles, always taking the largest leap possible when forwarding messages without passing the intended node
    - No redundant copies or processes
        - Based on the requirements of the assignment, we keep only one copy of a storage processes' data and maintain as few non-storage processes as possible so message handling can easily find the correct recipient