Failure Model for Chord Algorithm

Description

Our main aim of developing this model is to facilitate look up to accommodate any failure during the chord execution.

We have exceptions pertaining to the failures listed below-

1. **Crash failure** : A node halts, but is working correctly until it halts. The exception associated with it is –
   case class nodeCrashFailureException(smth:String) extends Exception(smth)

2. **Omission failure** :
   - Receive omission: A node fails to receive incoming messages. The exception associated with it is –

case class nodeRecieveOmissionFailureException(smth:String) extends Exception(smth)

   - Send omission: A node fails to send messages. The exception associated with it is –

case class nodeCSendOmissionFailureException(smth:String) extends Exception(smth)

3. **Timing failure** : A node's response lies outside the specified time interval. The exception associated with it is –
   case class nodeTimeoutFailureException(smth:String) extends Exception(smth)

4. **Response Failure**: The node's response is incorrect/ Arbitrary

Handling the Failures

Master actors watch over their workers, just as in real life, taking care if they are in problem while executing the work. Each actor defines its own supervisor strategy, which tells Akka how to deal with certain types of errors occurring in your children. We have implemented our OneForOneStrategy instead of sticking to the default strategy.

There are four possibilities after a failure happens

- Resume the node
- Restart the node
- Stop the node
- Escalate the node

In our model Master stops the node as the value contained during pushsum will be lost once it fails.

Challenges Faced

1. **Multiple nodes can fail simultaneously.** Handled it by,

   When a node ✣ fails, nodes whose finger tables include ✣ must find ✣'s successor. In addition, the failure of ✣ must not be allowed to disrupt queries that are in progress as the system is re-stabilizing. The key step in failure recovery is maintaining correct successor pointers, since in the worst case find predecessor can make progress using only successors. To help achieve this, each Chord node maintains a "successor-list" of its ✙ nearest successors on the Chord ring.

   If node ✣ no- tices that its successor has failed, it replaces it with the first live en- try in its successor list. At that point, ✣ can direct ordinary lookups for keys for which the failed node was the successor to the new successor. As time passes, stabilize will correct finger table entries and successor-list entries pointing to the failed node.

   After a node failure, but before stabilization has completed, other nodes may attempt to send requests through the failed node as part of a find successor lookup. Ideally the lookups would be able to proceed, after a timeout, by another path despite the failure.

   Optimized it by remembering what all nodes are no more into the system to avoid recomputations.

   State of each actor, i.e., whether it is alive or dead is maintained by the flag "isAlive". It is initialized to 1 whenever a node is created and upon failure of a node it is set to 0 and the node is killed.

Also maintained a max number of hops , to avoid infinite running

2. The random number generation of scala is not very random(needed in order output a set of distinct random nodes which fail.). Handled it by,
    - Written our own optimized random function using existing rand library.

Conclusion

Since the following output shows that we are able to do the look up inspite of node failures, our system is resilient
Sample

> run 5 5 3 // number of nodes number of lookups number of nodes to fail

Welcome to chord protocol system!!

You enetered numNodes =5

You entered numRequests = 5

the maximum number of nodes that can fail : 3

All files found successfully,system shutdown initiated! Please find summary below:

The total lookups done successfully is : 10

The total number of hops for all of them : 6

Average number of hops per request : 0.6

Awesome!!!!! Total time taken for the all the search is : 40 milliseconds

Limitations of Model

1. Lookup is definitely achieved but at the same time after a threshold of no of requests, dead letters become unavoidable
2. Based on the observations, we can conclude that this failure model implemented is very resilient.

References

1. http://doc.akka.io/docs/akka/snapshot/scala/fault-tolerance-sample.html#full-source-code-of-the-fault-tolerance-sample
2. http://danielwestheide.com/blog/2013/03/20/the-neophytes-guide-to-scala-part-15-dealing-with-failure-in-actor-systems.html
3. Section 5 of the paper uploaded on canvas