**Final Project – Option #1**

**General Map Reduce Framework**

**Urvi Patel**

**CSC 536**

**June 7, 2017**

**Running the App**

The purpose of this app is to take the MapReduce to a higher level of abstraction and allow it to work on a variety of jobs instead of just one.

This program may be run by opening 2 terminal windows, at the mapreduce2 directory, and running sbt in both. Once sbt has started up, type run in both windows. A menu will appear to run either the client or the server. In one terminal, select the client, option [1], and in the other select the server, option [2]. The mapreduce2 directory contains all the sbt build file, the code, configuration files, and resource files needed to run the app.

**Design**

The app is based on the mapreduce solution provided for homework #3, but provides a more general solution through the use of generics. There are 4 mappers and 4 reducers and this includes both remote and local mappers and reducers. There is also some fault tolerance built into this app via the SupervisorStrategy. It is very general however, and on any exception that is encountered, the failed actor will restart. Note that the OneForOne strategy is used so only the failed actor will be restarted. Also, a Helper class was created in the file messages.scala to hold the map, reduce, and aggregator functions for each job, as well functions to generate the input needed for each job.

The input job will need to be changed depending on which job is run. The code to create the MasterActor for the 3 jobs is in mapreduceclient.scala, lines 26 – 28. The following shows how to run the ReverseIndex job.

val master1 = system.actorOf(Props(classOf[MasterActor[String,String,String,Int,String]], inputA, helper.mapperA, helper.wordCountAgg, helper.reducerA), name = "master1")

val master2 = system.actorOf(Props(classOf[MasterActor[String,String,String,String,String]], inputB, helper.mapperB, helper.capWordAgg, helper.reducerB), name = "master2")

val master3 = system.actorOf(Props(classOf[MasterActor[String,String,String,String,Int]], inputC, helper.mapperC, helper.webReversalAgg, helper.reducerC), name = "master3")

master2 ! MAPME

master2 ! Flush

The MasterActor is a generic class that takes (in its constructor) the input, map function, aggregator function, and the reduce function. It then creates the ReduceActors which have the reduce function passed into the constructor. The routers are then passed into the MapActor constructor, along with the input, the map function and the aggregator function. Note the MapActor class must be generic as well. The MasterActor’s receive method merely sends a “MAPME” message to the MapActor. The MapActor in turn calls a (generic) run method, passing in the input, map function, and aggregator it received from the MasterActor. The map and aggregration phases are completed within the MapActor. The output of the aggregation phase is a HashMap and the MapActor then itereates over this HashMap and sends each (key, value) pair to the Reduce Actor. The Reduce Actor then applies the reduce function on each (key, value) pair and outputs the result.

**Minor Issues**

The first error seen was when trying to send the reduce function to the reduce actors. Apparently, the Java serializer does not work well with functions and the following error was given. Another student posted on the Google group that he also so this error when sending the reduce function to the ReduceActor.

[ERROR] [05/20/2017 09:18:41.187] [MapReduceClient-akka.remote.default-remote-dispatcher-14] [akka.tcp://[MapReduceClient@127.0.0.1:53526/system/endpointManager/reliableEndpointWriter-akka.tcp%3A%2F%2FMapReduceServer%40127.0.0.1%3A2552-0/endpointWriter](http://MapReduceClient@127.0.0.1:53526/system/endpointManager/reliableEndpointWriter-akka.tcp%3A%2F%2FMapReduceServer%40127.0.0.1%3A2552-0/endpointWriter)] Failed to serialize remote message [class common.ReduceIt] using serializer [class akka.serialization.JavaSerializer]. Transient association error (association remains live)

akka.remote.MessageSerializer$SerializationException: Failed to serialize remote message [class common.ReduceIt] using serializer [class akka.serialization.JavaSerializer].

I was able to get around this error by passing the reduce function definitions to the ReduceActor constructor so it would not need to be sent in a message.

Another issue occurred when trying to implement the MapReduce job for computing the reverse index proper names in a set of text files. I put the book title and URL in a file and read the file line by line, parsing for the URL and saving it in a variable called url. However, when tried to get the contents of the URl via the following:

var content = Source.fromURL(url).mkString

I received a MalformedURLException. The file was created in TextWrangler, as were the rest of the scala files so it was indeed a plain text file that didn’t have any special character. This line of code worked when the literal string was entered as an argument or when a literal string was saved into a variable in the program and then that variable is entered as an argument. However, when the URL was read from a file and save into a variable, the MalformedURLException was received. To get around this, a HashMap was created, with the book title as the key and the URL as the value.

**Pitfalls**

This greatest challenge was using generics to make this framework general. The first tactic was to define a case class with generic type in the messages.scala file, as follows:

case class MapIt[K, V, X, Y, U](input: HashMap[K, V], mapFn: (K, V) => List[(X, Y)], redFun: (X, List[Y]) => (X, U))

Then, in the receive method of the MapActor class, to match on this generic.

case MapIt[K, V, X, Y, U](input: HashMap[K, V], mapFn: (K, V) => List[(X, Y)], redFun: (X, List[Y]) => (X, U)) =>

// case code here

However, this gave a compile error, as shown below. Note, you will need to increase the zoom percentage on this document to 200% to see it clearly.

mapactor.scala:17: ‘)’ expected but ‘=>’ found.

[error] case MapIt[K, V, X, Y, U](input: HashMap[K, V], mapFn: (K, V) => List[(X, Y)], redFun: (X, List[Y]) => (X, U) =>

^

mapactor.scala:17: ‘)’ expected but ‘=>’ found.

[error] case MapIt[K, V, X, Y, U](input: HashMap[K, V], mapFn: (K, V) => List[(X, Y)], redFun: (X, List[Y]) => (X, U) =>

^

Next, the MapActor class was defined to be generic, but creating the actor using these generic types was not obvious. Also, the Master class needed to be parameterized as well since the input, map function, and reduce function needed to be sent from the MasterActor class. That is, the input, mapFn, and redFn were in the main app and needed to be sent to the MasterActor class.

class MapActor[K,V,X,Y,U](reduceActors: ActorRef, input:HashMap[K,V], mapFn: (K,V) => List[(X,Y)], redFn: (X, List[Y]) => (X,U)) extends Actor {

class MasterActor[K,V,X,Y,U](input:HashMap[K,V], mapFn: (K,V) => List[(X,Y)], redFn: (X, List[Y]) => (X,U)) extends Actor {

I tried to create MapActors as follows:

val mapActors = context.actorOf(RemoteRouterConfig(RoundRobinPool(numberMappers), addresses).props(Props(classOf[MapActor[K,V,X,Y,U]], input, mapFn, redFn,reduceActors)))

However, this gave the following error. I also saw some postings on the web using ClassTag, which is in the scala.reflection.\_ package. However, I could not get this working.

/Users/urvipatel/Desktop/PROJECT/Draft3\_WORDCOUNT\_CAPWORDS\_PAGERANK\_General/mapreduce2/src/main/scala/client/mapreduceclient.scala:16: type arguments [client.MasterActor] conform to the bounds of none of the overloaded alternatives of

[error]  value apply: [T <: akka.actor.Actor](creator: => T)(implicit evidence$2: scala.reflect.ClassTag[T])akka.actor.Props <and> [T <: akka.actor.Actor]()(implicit evidence$1: scala.reflect.ClassTag[T])akka.actor.Props

[error]   val master = system.actorOf(Props[MasterActor], name = "master")

[error]                               ^

After reading that you cannot serialize functions, I thought to try to pass the function (and input and map function) into the constructor for the MapActor. They are created in the main app so I tried to pass them to MasterActor and then to the MapActor. I made the MasterActor and MapActor generic classes.

class MapActor[K: ClassTag, V: ClassTag, X: ClassTag, Y: ClassTag, U: ClassTag](reduceActors: ActorRef, input:HashMap[K,V], mapFn: (K,V) => List[(X,Y)], redFn: (X, List[Y]) => (X,U)) extends Actor {

class MasterActor[K,V,X,Y,U](input:HashMap[K,V], mapFn: (K,V) => List[(X,Y)], redFn: (X, List[Y]) => (X,U)) extends Actor {

However, I couldn't figure out how to create the MasterActor object in mapreduceclient.scala.

val master = system.actorOf(Props[MasterActor], name = "master")

I did not know how to add parameters for a generic function to Props. I tried the following:

val master = system.actorOf(Props[MasterActor[String,String,String,Int,String]], inputA, helper.mapperA, helper.reducerA, name = "master")

However, I received the following errors:

/ mapreduce2/src/main/scala/client/mapreduceclient.scala:29: missing argument list for method mapperA in class Helper

[error] Unapplied methods are only converted to functions when a function type is expected.

[error] You can make this conversion explicit by writing `mapperA \_` or `mapperA(\_,\_)` instead of `mapperA`.

[error] val master = system.actorOf(Props[MasterActor[String,String,String,Int,String]], inputA, helper.mapperA, helper.reducerA, name = "master")

[error] ^

/mapreduce2/src/main/scala/client/mapreduceclient.scala:29: missing argument list for method reducerA in class Helper

[error] Unapplied methods are only converted to functions when a function type is expected.

[error] You can make this conversion explicit by writing `reducerA \_` or `reducerA(\_,\_)` instead of `reducerA`.

[error] val master = system.actorOf(Props[MasterActor[String,String,String,Int,String]], inputA, helper.mapperA, helper.reducerA, name = "master")

**Closer to Success**

It was suggested to create functions using the val keyword and after a bit of trial and error, I stumbled upon <https://gleichmann.wordpress.com/2010/11/08/functional-scala-functions-as-objects-as-functions/>, which had the proper syntax.

At this point, I was able to generate correct aggregator output, and the next step was to get this output into the Reduce Actor. I tried to do this via a case statement but was not successful:

var redPrepOutput = aggr(flattened)

reduceActors ! ReduceIt(redPrepOutput)

Where ReduceIt is a case class defined as follows, since I don't know the types:

case class ReduceIt[K, V](input: HashMap[K, List[V]])

I tried to match as follows:

case ReduceIt(m: HashMap[K, List[V]]) =>

//print statement

However, I got an error

reduceactor.scala:46: not found: type K

[error] case ReduceIt(m: HashMap[K, List[V]]) =>

reduceactor.scala:46: not found: type V

[error] case ReduceIt(m: HashMap[K, List[V]]) =>

I also tried using Futures, as described below, but was still unsuccessful.

I created a method getAggOutput() that returns a HashMap, in the MapActor

def getAggOutput: HashMap[X, List[Y]] = {

println("\*\*\*\*\* Inside getAggOutput, rMap is: " + rMap)

return rMap

}

Also, in the MapActor, the method is called that under the case object GETOUTPUT:

case GETOUTPUT =>

println("\*\*\*\*\* Inside GETOUTPUT")

getAggOutput

In the ReduceActor, I have the following:

implicit val timeout = Timeout(5 seconds)

val reducerInput = sender ? GETOUTPUT

val result = Await.result(reducerInput, timeout.duration)

println("\*\*\*\*\*\* RESULT is: " + result)

The print statements show that the value for rMap is correct:

\*\*\*\*\* Inside getAggOutput, rMap is: Map(lazy -> List(1), dog -> List(1, 1, 1, 1), jump -> List(1), fell -> List(1), s -> List(1), family -> List(1), belong -> List(1), over -> List(1), friend -> List(1), fox -> List(1, 1), tried -> List(1), man -> List(1), quick -> List(1), same -> List(1), brown -> List(1), best -> List(1))

However, I got the following error:

java.util.concurrent.TimeoutException: Futures timed out after [5 seconds]

And result was not assigned the value of the HashMap.

**Closing in on the Solution**

The solution to this issue was to change:

case GETOUTPUT =>

println("\*\*\*\*\* Inside GETOUTPUT")

getAggOutput

to

case GETOUTPUT =>

println("\*\*\*\*\* Inside GETOUTPUT")

**sender !**  getAggOutput

Now, the print statement shows that the HashMap in the result variable is indeed correct. However, when I tried to loop over the HashMap, as follows:

for( (key,value) <- result)

{

redOut += (key -> value)

}

I got the following error:

type mismatch;

[error] found : (Any, Any)

[error] required: (X, U)

[error] redOut += (key -> value)

I also tried:

for( (key,value) <- result)

{

redOut += (key.asInstanceOf[X] -> value.asInstanceOf[U])

}

However, I received the following error:

value withFilter is not a member of Any

[error] for( (key,value) <- result)

After all that, I decided to loop over the HashMap in the MapActor and send the (key, value) pair to the ReduceActor to apply the reduce function to that pair. This worked well for the WordCount and Web Link Reversal jobs. However, when running the ReverseIndex job, I received the following error.

[ERROR] [06/03/2017 23:30:56.465] [MapReduceClient-akka.remote.default-remote-dispatcher-14] [akka.tcp://MapReduceClient@127.0.0.1:59368/system/endpointManager/reliableEndpointWriter-akka.tcp%3A%2F%2FMapReduceServer%40127.0.0.1%3A2552-0/endpointWriter] Transient association error (association remains live)

akka.remote.OversizedPayloadException: Discarding oversized payload sent to Actor[akka.tcp://MapReduceServer@127.0.0.1:2552/remote]: max allowed size 128000 bytes, actual size of encoded class akka.remote.DaemonMsgCreate was 3798257 bytes.

After some quick Googling, I found that the setting is akka.remote.netty.tcp.maximum-frame-size. I set this to 30000000b in the application.conf file for the server and client configuration and the ReverseIndex job completed successfully.