Project Part-2: Map-Reduce

Description:

Define 2 classes, Mapper and Reducer, and other helper classes and put them in a jar - only for Java.

Job: will extend the Mapper class and implement the map() method. Similarly, it will extend the Reducer class and implement the reduce() method. This will be packaged as a jar. In C, this will be packaged as a .so.

In both cases, the assumption is that the job jar is available in the TT's classpath (LD_LIBRARY_PATH for C). Before running the job, you will have to manually copy the jar (or .so) to all the nodes running TT.

In Java, use Class.forName() to load the class. In C use dlopen() to load the .so; and dlsym() to invoke the map() and reduce() methods.

JobSubmitResponse jobSubmit(JobSubmitRequest)

JobStatusResponse getJobStatus(JobStatusRequest)

JobClient: This is invoked from command line with the following args: <mapName> <reducerName> <inputFile in HDFS> <outputFile in HDFS> <numReducers>

JobClient finds the location of JT using a conf file, and sends an RPC jobSubmit with the protobuf filled accordingly.

On success, the response contains the jobId.
The JobClient then sits in a while loop sending getJobStatus RPCs with the jobId until the JT responds with a completion response.

The client should print progress status to show percentage of map tasks started, and percentage of reduce tasks started on the console.

The JT assigns an id to the job, and queues the job for processing. It opens the file in HDFS, obtains the block locations to determine the number of map tasks required. It uses the locations to decide where to schedule the map tasks.

When a TT heartbeats, it uses the number of map/reduce slots available to decide if it can schedule tasks on the TT. The HB response contains information required to execute the map/reduce tasks.

The heartbeat from TT also contains information about the status of the tasks. It uses this information to respond to the getJobStatus RPC from the JobClient.

Once all the map tasks for a job are complete, it schedules the reducer(s) on the task tracker nodes based on availability.

HeartBeatResponse heartBeat(HeartBeatRequest)

Each TT heartbeats with the JT every second reporting information about how many map/reduce tasks it can run; and also providing status information about the current tasks it is running.

When the TT gets a map request, it places the request in an internal queue. The consumers of this queue are a fixed number of threads. The number of map slots sent by TT in the heartbeat request is the number of threads - number of map tasks already being executed.

Each thread picks up a MapTask from the queue. It then instantiates a class of type Mapper by loading the mapClass, and calling its constructor (empty constructor).

It then reads the appropriate HDFS block using readBlock, splits it into lines (\n terminated) and invokes the map() implemented by the class.

The map() will return a string that contains the records ((K,V) pairs) it wants to emit. The helper thread will write the record after adding \n to a file. This file is the output of the map task.

Once the input blocks are all processed, the map task is complete. The results are placed in another queue for the TT to pick up and provide status to the JT along with the name of the output file just created. Write the output of the map task in a file in HDFS. Give unique names to the map output files like job_<jobid>_map_<taskid>.

Once all the map tasks for a job are complete, the JT will schedule the reducers. To do this, it takes the outputs of the map tasks and divides them among all the reducers. It then schedules the reducers on the TTs.

When the tasktracker gets a reduce task, it queues it up like it does for map tasks.

A thread from the reducer thread pool picks up the reducer task and does the following.
- Read each HDFS file. This file contains the records that are emitted by the map task. To simplify, assume that the records as "\n" terminated.
- It invokes the reduce() method with each such line
- The reduce() returns a string. If it is non-null, the string is written as is to the output file

Once all the input files are processed, the reducer is complete and the result placed in another queue for the TT to pick up. The output of each reducer goes to the file <outputfile>_<jobId>_<reducerId>

The TT then communicates the status to the JT and the job is completed.

The final output can be read by reading all the files matching the name <outputfile>_<jobId>_*

Note: All required proto, java and IDL files have been attached