15-640 Distributed Systems

Lab3 – Map Reduce Engine Report

-Amey Ghadigaonkar

7/14/2014

**15 - 640 Distributed Systems**

**Lab 3 – Map Reduce Engine - Report**

**-Amey Ghadigaonkar.**

**Index:**

1. Major Assumptions
2. Cool Implemented Features of the project
   1. Low latency for launching mappers/reducers through pre-launched processes
   2. Failure tolerance for datanodes during initialization through task queues
   3. Good CPU utilization and Low Dispatch latency using multiple processes
   4. Smart reduction phase using key value partitioning
   5. Virtual file system implemented using tree structure
   6. Jobs can be initialized at any node except for namenode
   7. Clean I/O for MR jobs and communication
   8. Configuration file
3. Features not implemented or to be improved if given more time
   1. **Smarter** Job Scheduler
   2. Global Conf File located on shared resource (e.g. andrew)
   3. Multiple namenodes / a committee of masters
   4. Combiner

**TODO: stop\_JVMs**

1. **Major Assumptions**

* Ialways have **at least 3 Datanodes in addition to a main Namenode**. Otherwise, the system won’t function as expected since the write quorum is not fulfilled. In such a circumstance, the admin/user should ensure that there are at least 2 datanodes and restart the system as shown in part <?> of this report
* Multiple datanodes can exist on the same machine, **but each datanode must have a different root folder** (as I show in steps to deploy the system for sys admins). If they have the same root node, they will overwrite each other’s data leading to inconsistent state.
* The main namenode is always available. If the namenode fails, then there is no recovery line.
* No MR jobs can be run at master. I have taken this design decision to avoid any overhead for the namenode as it is the controller for other nodes.

**(Imp: windows uses “.” While unix uses “/”. This manual is created for AFS i.e. unix. For example “bin.client.Client” vs. “bin/client/Client”)**

1. **COOL FEATURES OF THE PROJECT:**
   1. Low latency for launching mappers/reducers through pre-launched processes

I have maintained a separate process for each mapper/reducer. Also, these pre-launched processes are pre-initialized on each datanode. I maintain an array of these processes in case they stop responding so that I can destroy these processes and initialize new processes in their place.

* 1. Failure tolerance for datanodes during initialization through task queues

I have maintained addition and deletion queues for file transfer. Each task is a block of the file to be added/deleted from a datanode. Tasks are removed from queues once their acknowledgement is received. Thus, it results in a failure tolerant model.

* 1. Good CPU utilization and Low Dispatch latency using multiple processes

I try to utilize CPU processing power by maintaining multiple pre-launched processes as mentioned in point a. The number of running processes depends on the number of CPU cores. I take this information from “/proc/cpuinfo” on Andrew servers. If running on Windows/other machines, I assume it to be 6.

* 1. Smart reduction phase using key value partitioning

My scheduler first takes 12\*log(n) rows from each mapper and perform map operation, where n is the number of rows in the file. I remember this number from my undergrad algorithms course. I don’t know the exact implications of this number, but it should give a good key set for the next step.

Next, I take the keys from this step and send it to namenode, which is responsible for partitioning the overall reduce jobs so that keys in a particular range are sent to one particular reducer. This would ensure good performance and eliminate the need for complex, multi-staged reduce.

* 1. Virtual file system implemented using tree structure

My virtual filesystem stores directory and file information in a hierarchical structure like a normal operating system instead of a flat structure. This enables efficient lookup of contents and also allows file system operations like “ls” and “mkdir”.

* 1. Jobs can be initialized at any node except for namenode

Jobs can be initialized at any node except the main namenode (the controller node). I have taken this design decision to avoid any overhead for the namenode as it is the controller for other nodes.

* 1. Clean I/O for MR jobs and communication

My design emphasizes on the fact that the allocator of a resource must also be responsible for its de-allocation. This leads to a clear interface for communication in the “Communicator.java” class and “filesystem” package.

* 1. Configuration File

The configuration file is present in the Constants.java file inside ‘conf’ package.

1. **FEATURES TO BE IMPROVED IF GIVEN MORE TIME:**

1. Global Config file

I would have liked to implement a global config file which would be accessible to all nodes during initialization. Although this feature was not particularly hard t implement, I simply did not get time to implement it. It would also serve as a rough communication mechanism between operating nodes.

1. Smarter Job Scheduler

At the moment, the scheduler considers primarily the availability of files, and then the number of usable pre-launched processes available on the datanodes for scheduling jobs. I would have liked to include the expected runtimes of the mapper/reducers into the scheduling algorithm. As described in part 2.c., I execute sample Map job, and I store the initial and final sizes of the map phases. If I add in a timer for the tasks, I can also estimate the time needed for the map phase. I can use this data to make my MR scheduler smarter.

1. More power to the System Admin

I would have liked to include the functionality for the system admin to explicitly select which datanodes out of all the registered datanodes will be available for the application programmer to use. This could be an extension to the already existing configuration file.

1. Multiple namenodes / a committee of masters

I have not thought about how to do this. But it would definitely be a problem worth solving, and it looks interesting too!

1. Combiner

A combiner would have been cool to have, since it would have efficiency of mapper output files tremendously. If I would have had a team mate, then I would have definitely tried this feature.

1. Filesize thread not working on Andrew

My file size thread which calculates the size of files stored on a particular datanode, does not work as expected on Andrew (maybe because it’s a DFS) but works on my Windows machine. I would have liked to correct this problem for Andrew.