

Hadoop 0.2.0 MapReduce Developer doc

@author Sanjay Dahiya – sanjayd@yahoo-inc.com

MapReduce flow starts with `JobClient.runJob(JobConf jobConf)`. `jobConf` is the job configuration that also maps to the job config xml file. Key operations in this flow follow.

- Create a `JobClient` object, pass `jobConf`.
- `JobClient` constructor initializes `LocalJobRunner` or `JobTracker` depending on “*mapred.job.tracker*” attribute pointing to ‘local’ or any remote host in conf. This document follows `JobTracker`.
- Invoke `jobClient.submitJob(jobConf)` with same `jobConf` object again. ***[Why pass the same object again? API design issue]***
- `jobClient.submit()`
 - gets the `FileSystem` details from `jobConf` and copies the user supplied jar file containing MapReduce implementation to DFS. Also sets replication properties.
 - Serializes the `jobConf` object in XML and copies the XML file to DFS.
 - Call `JobTracker.submitJob(submitJobFile)`, `submitJobFile` is path to the `job.xml` file in DFS.
- `JobTracker.submitJob()`
 - Create `JobInProgress`(see next) object for the job, pass it the Path to `job.xml` in DFS.
 - `JobInProgress` Creation
 - Copy `job.xml` file and Jar file containing MapReduce implementation from DFS to local FS.
 - Read job attributes from `jobConf` and set them in object properties.
 - Add `JobInProgress` object to housekeeping data structures – jobs Map and jobsByArrival Vector.
 - Add `JobInProgress` object to `jobInitQueue`, and call `notifyAll` on the queue.
 - `JobTracker` startup also starts a thread `JobInitThread`
 - that waits on `jobInitQueue`, `notifyAll()` causes this thread to wake up and read from the queue.
 - `JobInitThread` reads a new job from the `jobInitQueue` (this job is an instance of `JobInProgress`) and calls `initTasks()` on this job.
- `JobInProgress.initTasks()`
 - Deserialize local `job.xml` to `JobConf` ***[Why de-serialize, that was done in constructor already?]***.
 - Get the `InputFormat` class name from conf and load it using `URLClassLoader` from the Jar file, which is now on local disk. It can be supplied by user or be a part of hadoop. We assume we are using `TextInputFormat` for this document. ***[JobInProgress is not using the InputFormat as the split is done based on file size, why copy the jar here? Do we want to allow extending getSplits() by users ?]***

- Input to MapReduce needs to be divided into splits to pass to different Map tasks. InputFormat.getSplits() does this.
- InputFormatBase.getSplits()
 - List all files that are part of input data (like ls -R).
 - Get total size of these files and get the size of one split by totalSize/numSplits, numSplits is passed in jobConf.
 - If any of the input files is not available now exit without executing MapReduce [***Sure we want to do this ?***].
 - FileSplit is a reference to part of a file, it contains File Path in DFS and start and end offsets in the file. A Split size is governed by minSplitSize and maxSplitSize attributed in jobConf.
 - Files smaller than the split size are considered as single splits.
 - [***return doesn't need to create a array of splits.size() for typecasting, a zero size array should be good enough***].
- Sort splits in decreasing order of size [***not clear why?***]
- For each split create a new Map task – TaskInProgress object. TaskInProgress is used for both Map and Reduce Tasks. Pass the assigned FileSplit to the task.
 - TaskInProgress() creation assigns unique id to the task.
 -

All tasks exist in JobTracker's Queue till they are fetched by TaskTrackers for execution on respective nodes.

TaskTracker –

TaskTracker runs on worker nodes, it takes Map and Reduce tasks from JobTracker, executes them and sends status back to JobTracker.

TaskTracker.main() -> TaskTracker.run()

- Initialize with address of TaskTracker.
- while (running) call offerService() – basically poll TaskTracker for new Tasks and send status of running tasks.
- After every HEARBEAT_INTERVAL do following
 - Get status of all running tasks and create a TaskReport. Send to TaskTracker.
 - If currently running map tasks are less than mapTotal get more tasks from TaskTracker. [***Starting only one task here ?***]
 - Starting a new Task
 - Create new TaskInProgress add to runningTasks Map. This task can be a Map task or a Reduce Task.
 - Start the Task with TaskInProgress.launchTask()
 - Copy Task Jar file from DFS to local – localizeTask()
 - Depending on Map /Reduce Task create – MapTaskRunner / ReduceTaskRunner.
 - Start the TaskRunner thread. Job of this thread is to prepare and exec a new process for the Task. After exec this Thread reads stdout from new process and is active till the process is running. The new process runs TaskTracker.Child.main()

- Extract the previously downloaded Jar file and add to class path [*why extract ? jst adding class jar should do, unless there is something like a shared lib in it*].
 - New process takes job id and TaskTracker id/port as arguments.
 - Kill tasks that we havent heard from recently.
- TaskTracker.Child.main()
 - Gets a task id to run.
 - TaskTracker maintains a Map of all tasks(TaskInProgress), get the assigned taskid.
 - Get the job conf file and create a JobConf object, [*not sure whats happening here with addFinalResource()*]
 - Set working directory [*Using static call on FileSystem, will this conflict if there are multiple Map tasks ?*]
 - Call MapTask.run() to run the assigned Map task.
- MapTask.run()
 - Create SequenceFile.writer[noOfReduceTasks], to collect output from the map tasks.
 - Create OutPutCollector with above writers, each call to output.collect() appends to one of the writers, writer selection is done to distribute appends evenly, see HashPartitioner.
 - Use CombiningCollector if in jobConf (*not covered in this doc yet*)
 - Use InputFormat.getRecordReader(), which was loaded using URLClassLoader earlier, and pass it the assigned FileSplit for this MapTask. This record reader can now read records from the assigned split from DFS using RecordReader.next().
 - [*An anonymous inner class is created to wrap reader and add reporting functionality to it, not sure if that's the right way to do it*]
 - Create a MapRunnable object (use class passed in JobConf) and call run, with input reader, outputCollector.
- MapRuner.run()
 - Get next key and value to from input reader and call user supplied Mapper.map().