# HotTopics in BigData

Kyungyong Lee

# Announcements

- Homework
  - https://docs.google.com/document/d/1M1GY5Q7TP8Xo-GD0Nh4YPdGUoOaiNEzZ5L2IFGZXgt8/edit?usp=sharing
  - Also uploaded in the ecampus
  - Due: April 15th
  - Submission to ecampus
- Exam
  - April 4$^{th}$ at the class time
  - 50 mins.

# Hadoop MapReduce Job Commands

- Create a directory to store output class files
  - mkdir wordcount_classes
- Compile MapReudce job
  - javac -classpath /usr/local/hadoop/share/hadoop/common/hadoop-common-2.8.0.jar:/usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.8.0.jar -d wordcount_classes  WordCount.java
- Create a jar file using .class files
  - jar -cvf WordCount.jar -C wordcount_classes/ .
- Running a MapReduce job
  - hadoop jar WordCount.jar kr.ac.kookmin.cs.bigdata.WordCount /home/${ID}/reviews_sample.txt /home/${ID}/output

# Check Hadoop Job Status

- Get a list of jobs
  - mapred job –list
- Get a status of a job
  - mapred job –status {job-id}
- Kill a running job
  - mapred job –kill {job-id}
- More examples
  - https://hadoop.apache.org/docs/r2.6.3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredCommands.html#job

# Hadoop Input Formats

- InputFormat
  - An interface to parse input dataset to be passed to Mapper function
  - *TextInputFormat* – Treats each '₩n'-terminated line of a file as a valu e (line-based records, logs)
  - *KeyValueTextInputFormat* – Maps '₩n'- terminated text lines of "k SE P v"
  - *SequenceFileInputFormat* – User-defined (K,V) format for Hadoop sp ecific high performance format

# Hadoop Output Formats

- TextOutputFormat
  - Default - writes “key val₩n” strings to output file
  - Can be read by KeyValueInputFormat
  - Human readable
- SequenceFileOutputFormat
  - Uses a binary format to pack (k, v) pairs
  - Usually an input for another MR task (intermediate result)
- NullOutputFormat
  - Discards output
  - Only useful if defining own output methods within reduce()

# Locality Aware Execution

- Avoid network usage for block delivery
- Each block is replicated in three locations – by default
- If a TaskTracker that contains a given block is busy, the JobTracker searches other TaskTrackers that contain the replica
  - If there is, assign to the TaskTracker
  - Otherwise, select a free TaskTracker that is close to the current replica
- Late scheduling
  - How about waiting extra time hoping the TaskTrackers containing the block become idle before assigning to other TaskTrackers

# Fault-Tolerance of Hadoop

- Node (TaskTracker) failure
  - Heartbeat message between JobTracker and TaskTracker to detect no de failure
  - MAKE IT SIMPLE: reassigning the jobs of failed host to other healthy nodes
  - Failure during Map Tasks
    - Generally need to re-execute all Map tasks (even succeeded ones) by the faile d host
  - Failure during Reduce Tasks
    - No need to re-execute already completed ones

# Fault-Tolerance of Hadoop

- Speculative execution
  - A task that is executing slowly comparing to other tasks
    - Resource (CPU, memory paging) competition
    - System/host error
  - Map and Reduce tasks are idempotent
  - Task re-execution policy (Once active research topic)
    - Task completion ratio
    - If task is last few Map or Reduce tasks

# Fault-Tolerance of Hadoop

- Master (JobTracker) failure
  - Checkpointing the current job status is a viable approach - cf. FSIma ge and transaction log in the HDFS
  - In reality, a client has to rerun it – **SIMPLICITY**
  - It is a single node – remember that as the number of machines incre ases, the number of failure also increases
  - Master needs special care

# MapReduce Task Granularity

- The number of Map tasks : M
- The number of Reduce tasks : R
- The number of hosts in a MapReduce cluster : H
- M and R >> H
  - Good for load balancing
  - Good for recovery from machine failure
- M is determined by the number of blocks
- R is configurable by a user
- Each Reduce task output is written into separate file – part-xx xxx, xxxxx is task ID

# Setup and Cleanup Function

- void setup(Context context)
  - Called once at the beginning of the task (per block)
  - Preparing a shared dataset to conduct Map task (ex. Ignore word list)
- void cleanup(Context context)
  - Called once at the end of the task(per block)
  - Cleanup temporary folder if it was created during Map/Reduce

# Multiple MapReduce Job in a Driver

- Map1 → Reduce1 → Map2 → Reduce2 → Map3 → …
- A large number of solutions to provide such features
  - Sequentially run jobs with job.waitForCompletion(true)
  - Other various methods exist
    - Oozie, luigi, Crunch, Cascading, …

```
@Override
public int run(String[] args) throws Exception {
    Job job1 = Job.getInstance(getConf());
    job1.setJarByClass(Job1.class);
    .........
    job1.waitForCompletion(true);

    Job job2 = Job.getInstance(getConf());
    job2.setJarByClass(Job2.class);
    .........
    job2.waitForCompletion(true);

    return 0;
}
```

# Setting Number of Reduce Tasks

- The number of Map tasks is generally determined by the number of blocks
- Too small number of Reduce tasks
  - Overhead to the Reduce task – Be cautious for the out of memory
- Too many number of Reduce tasks
  - Task scheduling overhead
  - Output file write overhead – note that each Reduce task creates part-r-xxxxx file
- Generally, it is advisable
  - Set the number of reduce tasks small that does not cause out-of-memory issue and it does not take too long (trial and error)

# Setting the Number of Reducers

- Command line arguments
  - -Dmapred.reduce.tasks=number of reduce tasks
  - EX: hadoop jar WordCount.jar kr.ac.kookmin.cs.bigdata -Dmapred.reduce.tasks=2 /home/${ID}/reviews_sample.txt /home/${ID}/output
- Set as a job configuration in source code
  - Job.setNumReduceTasks(2);