Lecture 25
Hadoop
Hadoop

• An open-source implementation of MapReduce
• Design desiderata
  – Performance: support processing of huge data sets (millions of files, GB to TB of total data) by exploiting parallelism, memory within computing clusters
  – Economics: control costs by using commodity computing hardware
  – Scalability: a larger the cluster should yield better performance
  – Fault-tolerance: node failure does not cause computation failure
  – Data parallelism: same computation performed on all data
Cluster?

- Hadoop is designed to run on a cluster
  - Multiple machines, typically running Linux
  - Machines connected by high-speed local-area network (e.g. 10-gigabit Ethernet)

- Hardware is:
  - High-end (fast, lots of memory)
  - Commodity (cheaper than specialized equipment)
Principles of Hadoop Design

• *Data is distributed* around network
  – No centralized data server
  – Every node in cluster can host data
  – Data is *replicated* to support fault tolerance

• *Computation is sent to data*, rather than vice versa
  – Code to be run is sent to nodes
  – Results of computations are aggregated as tend

• *Basic architecture is master/worker*
  – Master, aka JobNode, launches application
  – Workers, aka WorkerNodes, perform bulk of computation
Components of Hadoop

• MapReduce
  – Basic APIs in Java supporting MapReduce programming model

• Hadoop Distributed File System (HDFS)
  – Applications see files
  – Behind the scenes: HDFS handles distribution, replication of data on cluster, reading, writing, etc.
Hadoop Execution: Startup

1. MapReduce library in user program splits input files into pieces (typically 16-64 MB), starts multiple copies of program on cluster

2. One copy is master; rest are workers. Work consists of map, reduce tasks

3. Master keeps track of idle workers, assigns them map/reduce tasks

[Discussion adapted from Ravi Mukkamala, “Hadoop: A Software Framework for Data Intensive Computing Applications”; Hadoop 1.2.1 “MapReduce Tutorial”]
Hadoop Execution: Map Task

1. Read contents of assigned input split
   Master will try to ensure that input split is “close by”
2. Parse input into key/value pairs
3. Apply map operation to each key/value pair; store resulting intermediate key/value pairs on local disk
4. File is sorted on output key, then partitioned based on key values
5. Locations of these files forwarded back to Master
6. Master forwards locations of files to relevant reduce workers
   - Which reduce workers get which files depends on partition
   - Partition assigns different key values to different reduce tasks
Hadoop Execution: Reduce Task

1. Fetch input (files produced by map tasks and sent by master)
2. Sort input data by key
3. For each key, *apply reduce operation* to key / list of values associated with key
4. Write result in file (one output file / key, often, but configurable)
5. Return location of result files to Master
Configuring MapReduce Execution

• Many configuration parameters to tune performance!
  – Number of maps
  – Number of reduces
  – Splitting of input
  – Sorting, partitioning
  – Etc.

• Hadoop MapReduce tutorial gives a starting point
  https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
Fault-Tolerance

• Big clusters = increased possibility of hardware failure
  – Disk crashes
  – Overheating
• Worker failure
  – Master pings worker periodically: no response = worker marked as failed
  – Tasks assigned to failed worker added back into task pool for re-assignment
  – This works because *functional nature* of MapReduce ensures no shared state, while HDFS ensures *data is replicated* (so data hosted by failed node is still available)
• Master failure
  – Masters write checkpoint files showing intermediate progress
  – If master fails, a new master can be started from the last checkpoint
  – In practice: job generally restarted
Setting Up Hadoop

• Three possibilities
  – Local standalone (everything run in one process)
  – Pseudo-distributed (tasks run as separate processes on same machine)
  – Fully distributed (cluster computing)

• ???
  – Standalone usually used for development, debugging
  – Pseudo-distributed to analyze performance bottlenecks
  – Fully-distributed used for final job runs
Installing, Running Stand-alone Hadoop

• Download 0.20.2, extract files (remember where!)
  – There are multiple “development streams” in Hadoop: 0.xx, 1.xx, 2.xx
  – Why? Who knows
  – 0.20.2 is one guaranteed to work with Windows / Macs / etc.

• Follow instructions in P5 (basically, need to tell Eclipse where Hadoop libraries are that you just downloaded)
  – Download Hadoop-0.20.2.tar.gz
  – Unpack
  – Add .jar files listed in P5 to build path
Writing a Hadoop Application

• MapReduce
  – One class should extend Mapper<K1,V1,K2,V2>
    • K1, V1 are key/value classes for input
    • K2, V2 are key/value classes for output
  – Another should extend Reducer<K2,V2,K3,V3>
    • K2, V2 are key/value classes for inputs to reduce operation
    • K3, V3 are output key/value classes

• Main driver
  – Need to create a Job (Hadoop class) containing configuration settings for job
  – Settings include input / output file formats for job, input file-slice size, key/value types, etc.
Running a Hadoop Application

- Arguments configuring job given as command-line arguments
  - To parse these arguments, can use GenericOptionsParser class provided by Hadoop.
  - Can also “roll your own” command-line parser, but Hadoop will give a warning
- Need to have “chmod” installed in order to run!
  - Unix command for changing permissions on files and directories.
  - This command is not native to Windows!
  - Easiest way to obtain a chmod.exe:
    - Install Cygwin (“Unix for Windows”)
    - In Cygwin installation, locate directory containing “chmod.exe” (in my installation, it is “C:\cygwin\bin”).
    - Add this directory to your Windows path by editing Path system variable
      - Right-click on Computer, select “Properties”
      - Select “Advanced system settings”
      - Click on “Environment Variables”
      - Select “Path” variable in “System variables” pane, then Edit… button
      - Go to end, add “;” then path where chmod.exe is