Map - Reduce

Distributed Systems: 15-440/640
Purpose of this Session

- Crash-Course on Hadoop and Quick-Review of Concepts
- Quick-look at the Big-Components
- Checklist of the “must-haves” for this project
- Answer some FAQs asked during the TA office hours
- General expectations
- Question and Answers
Map-Reduce Overview - Hadoop

Figure 6-1. How Hadoop runs a MapReduce job
DFS Overview - HDFS

Flow Diagram:
- Client sends Job (jar file) to Master Node.
- Master Node distributes the job to Slave Nodes.
- Slave Nodes execute the tasks and send output to Master Node.
- Master Node collects the outputs and distributes them back.
- Data is replicated across Slave Nodes for redundancy.

Key Components:
- Client
- Master Node
  - Job Tracker
  - Slave Nodes
    - Task Tracker
    - Task
    - HDFS
DFS Overview - HDFS

MapReduce layer

HDFS layer

multi-node cluster
Map-Reduce Operational Overview

Note: You may have more than one output file (corresponding to each reducer)
Motivations for Map-Reduce

- Big-Data processing
- Increased utilization of commodity infrastructure
- Similar things can be done in parallel :)
- Scaled-Out, Distributed
- Fault-Tolerant, Replicated
- Hip and Cool !!! Really
## Elements of Concern - Architecture

<table>
<thead>
<tr>
<th>Communication</th>
<th>Information</th>
<th>Interaction (User Experience)</th>
<th>Computation/Processing</th>
<th>Failure Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Master a.k.a Job Tracker</td>
<td>Information about the chunks/splits of the file</td>
<td>Ability to submit a job</td>
<td>Common structure to take mapper and reducer code of the application programmer</td>
<td>Auto-replicate of file chunk if the node fails to maintain the RF</td>
</tr>
<tr>
<td>Local Master a.k.a Task Tracker</td>
<td>Information about the nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status and Health Checking of i. Nodes and ii. Map Tasks iii. Reduce Tasks</td>
<td>Global State: i. Status of Map Tasks ii. Status of Reduce Tasks iii. Status of Job</td>
<td>Ability to pull print about the job</td>
<td>Automatically send the his source-code across the nodes where file-chunks are available</td>
<td>Re-start the map/reduce task if one fails</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Threshold on re-attempts</td>
</tr>
<tr>
<td>Scheduling the tasks appropriately</td>
<td>Local State: i. Status of the Mapper ii. Status of Reducer iii. Intermediate files and I/O handling</td>
<td>Put yourself in the system admin’s shoes who is managing this cluster</td>
<td>Data-Locality knowledge for the master Schedule some work as soon as something slot is free</td>
<td>Status reporting if a job failure</td>
</tr>
<tr>
<td>Splitting the input file into chunks (Data actually needs to be sent and replicated)</td>
<td></td>
<td></td>
<td>How many slots?</td>
<td>Capturing and providing as much as relevant information to application programmer</td>
</tr>
</tbody>
</table>
Required Deliverables

- Distributed File System
- Map reduce framework
- Administration/Bootstrapping tool or instructions
- 2 Examples to showcase your framework
- Detailed Report
Distributed File System

- Don’t complicate things!

- Think of a way to:
  - Split large files into chunks
  - Transfer the chunks to different hosts
  - Track where chunks exist.
  - Maintaining a replication factor for fault tolerance
Distributed File System (Cont.)

Important:

- Use of afs after initial bootstrapping for your framework is **not** allowed!

- For all purposes think of it as each node having their own local storage communicating through your dfs framework.
Map-Reduce Framework

- Hadoop's Map-Reduce Architecture - a good starting point.
- Refer the Word Count Example
- The Same map reduce code works on all file chunks on same/different hosts
- Maximize efficiency by running jobs on locally available files
Map-Reduce Framework (Cont.)

Important:

● Fault Tolerance is Expected.
  What happens
  ○ if mapper fails?
  ○ if reducer fails?

● Config file should contain all the parameter your framework can support.
  ○ Some essential parameters like Input format, output format, input path, output path, mapper class and reducer class are expected
What you don't need to do

- You do *not* need to exactly emulate Hadoop or HDFS.
- No need for consistency, conflict resolution etc in your DFS.
Report checklist

“Extremely important part of the project”. Much more important than you think!

- Things we are looking for here:
  - Overall design and implementation of the MapReduce framework
  - Design and implementation of the distributed framework
  - Programmer API
  - Replica creation
  - Interaction between MapReduce and distributed file system
  - Work conservation and data location awareness
  - Launching and relaunching of Mappers and Reducers
  - Various Tradeoffs in your design.
  - **Build instructions.** If these are missing, we can’t do much. Please do not give us just jars, we need to be able to build your project while testing. Also, do not give us just jars of your example. We need the source code for those too.

“The Submitted code must work on afs. Please test it in afs before submitting”
Common Questions

- Do we need to merge output of reducers?
  - No, 1 output per reducer.
- Do we need to build a distributed file system? If yes, how much?
  - Refer previous decks
- Hey hadoop does it this way, is that what we are supposed to do?
  - No, You are welcome to use your own design.
- Fixed records - what does that mean?
  - Simplest case - Line of string.
- Do I need a shell on each node?
  - No, its up to your design
References