Hadoop Map Reduce
MapReduce

- 2-in-1
  - A programming paradigm
  - A query execution engine
- A kind of functional programming
- We focus on the MapReduce execution engine of Hadoop through YARN
Overview

Developer → MR Program → Driver

MR Job → Master node → Slave nodes
Job Execution Overview

- Driver
- Job submission
- Job preparation
- Map
- Shuffle
- Reduce
- Cleanup
Job Submission

- Execution location: Driver node
- A driver machine should have the following
  - Compatible Hadoop binaries
  - Cluster configuration files
  - Network access to the master node
- Collects job information from the user
  - Input and output paths
  - Map, reduce, and any other functions
  - Any additional user configuration
- Packages all this in a Hadoop Configuration
# Hadoop Configuration

<table>
<thead>
<tr>
<th>Key: String</th>
<th>Value: String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>hdfs://user/eldawy/README.txt</td>
</tr>
<tr>
<td>Output</td>
<td>hdfs://user/eldawy/wordcount</td>
</tr>
<tr>
<td>Mapper</td>
<td>edu.ucr.cs.cs226.eldawy.WordCount</td>
</tr>
<tr>
<td>Reducer</td>
<td>…</td>
</tr>
<tr>
<td>JAR File</td>
<td>…</td>
</tr>
<tr>
<td>User-defined</td>
<td>User-defined</td>
</tr>
</tbody>
</table>

- Master node
- Serialized over network
Job Preparation

- Runs on the master node
- Gets the job ready for parallel execution
- Collects the JAR file that contains the user-defined functions, e.g., Map and Reduce
- Writes the JAR and configuration to HDFS to be accessible by the executors
- Looks at the input file(s) to decide how many map tasks are needed
- Makes some sanity checks
- Finally, it pushes the BRB (Big Red Button)
Job Preparation

Master node

Configuration

HDFS

InputFormat#getSplits()

FileInputSplit
Path Start End

JAR File

Split$_1$, Split$_2$, .., Split$_M$

Mapper$_1$, Mapper$_2$, .., Mapper$_M$
Map Phase

- Runs in parallel on worker nodes
- M Mappers:
  - Read the input
  - Apply the map function
  - Apply the combine function (if configured)
  - Store the map output
- There is no guaranteed ordering for processing the input splits
Map Phase

Master node

IS₁  IS₂  IS₃  IS₄  IS₅  …  ISₘ
Map Task

- Reads the job configuration and task information (mostly, InputSplit)
- Instantiates an object of the Mapper class
- Instantiates a record reader for the assigned input split
- Calls Mapper#setup(Context)
- Reads records one-by-one from the record reader and passes them to the map function
- The map function writes the output to the context
MapContext

- Keeps track of which input split is being read and which records are being processed
- Holds all the job configuration and some additional information about the map task
- Materializes the map output
Map Output

What really happens to the map output?

It depends on the number of reducers

- 0 reducers: Map output is written directly to HDFS as the final answer
- 1+ reducers: Map output is passed to the shuffle phase
Shuffle Phase

- Executed only in the case of one or more reducers
- Transfers data between the mappers and reducers
- Groups records by their keys to ensure local processing in the reduce phase
Shuffle Phase

Map_1 → Reduce_1
Map_2 → Reduce_2
Map_3 → Reduce_2
... → ...
Map_M → Reduce_N
Shuffle Phase (Map-side)
Shuffle Phase (Reduce-side)
Reduce Phase

- Apply the reduce function to each group of similar keys
Output Writing

- Materializes the final output to disk
- All results are from one process (mapper/reducer) are stored in a subdirectory
- An OutputFormat is used to
  - Create any files in the output directory
  - Write the output records one-by-one to the output
  - Merge the results from all the tasks (if needed)
- While the output writing runs in parallel, the final commit step runs on a single machine
MapReduce Examples

- Input: A log file
- Filter
- Aggregation
- Conversion
Advanced Issues

- Map failures
- Reduce failures
- Straggler problem
- Custom keys and values
- Efficient sorting on serialized data
- Pipeline MapReduce jobs