High Performance Computing with Hadoop
WV HPC Summer Institute 2014

E. James Harner
Director of Data Science

Department of Statistics
West Virginia University

June 18, 2014

Outline

Overview

Hadoop is a framework for parallel processing: decompose a problem into independent units of work, process, and recombine. It is an open-source framework for large-scale data storage and distributed computing, built on the MapReduce model.

Used in conjunction with HDFS (Hadoop Distributed Filesystem).

Used for extract-transform-load (ETL), image processing, data analysis, etc. It is useful for big data and compute intensive tasks.

MapReduce Primer

Two phases: *Map* and *Reduce*

Map Phase:

1. Each cluster node takes a piece of the initial mountain of data and runs a Map task on each record (item) of input.
2. The Map tasks all run in parallel, creating a key/value pair for each record. The key identifies the item's pile for the reduce operation. The value can be the record itself or some derivation thereof.

The Shuffle: At the end of the Map phase, the machines all pool their results. Every key/value pair is assigned to a pile, based on the key.

Reduce Phase:

1. The cluster machines then switch roles and run the Reduce task on each pile. The Reduce task gets the entire pile (that is, all of the key/value pairs for a given key) at once
2. The Reduce task typically, but not necessarily, emits some output for each pile

Calculate Average Call Length for Each Date

Map task

- Receives a single line of input (that is, one input record)
- Uses text manipulation to extract the {date} and {length} fields
- Emits key: {date}, value: {length}

Reduce task

- Receives key: {date}, values: {length1 . . . lengthN}, i.e., each reduce task receives all of the call lengths for a single date.
- Calculates the mean (divides the total call length by the number of calls)
- Outputs the date and the mean call length

Number of Calls by Each User, on Each Date

Map task

- Receives single line of input
- Uses text manipulation to extract {date}, {caller num}
- Emits key: {date} {caller num}, value: 1

Reduce task

- Receives key: {date} {caller num}, value: {1 . . . 1}
- Loops through each item, to count total number of items (calls)
- Outputs {date}, {caller num} and the number of calls

Run Algorithm on Each record

Map task

- Receives single line of input
- Uses text manipulation to extract function parameters
- Passes those parameters to a potentially long-running function
- Emits key: {function output}, value: {null}

Reduce task

None

Binary and Whole-file Data

By default, when you point Hadoop to an input file, it will assume it is a text document and treat each line as a record. There are times when this is not what you want: e.g., performing feature extraction on sound files, or performing sentiment analysis on text documents. How do you tell Hadoop to work on the entire file, be it binary or text?

The answer is to use a special archive called a SequenceFile. A SequenceFile is similar to a zip or tar file, in that it's just a container for other files. Hadoop considers each file in a SequenceFile to be its own record.

Use Java API or use `forqlift`.

Compute Solutions

- AWS: Build your cluster using virtual servers on Elastic Compute Cloud (EC2)
- AWS: Leverage the Hadoop-on-demand service called Elastic MapReduce (EMR)
- myHadoop: WVU High Performance Computing `https://github.com/glennklockwood/myhadoop/`

`myHadoop` provides a framework for launching Hadoop clusters within traditional high-performance compute clusters and supercomputers. It allows users to provision and deploy Hadoop clusters within the batch scheduling environment of such systems with minimal expertise required.

Hadoop Strategies

Submit work to a cluster:

- Streaming: Write the Map and Reduce operations as R scripts, or any other scripting language. The Hadoop framework launches your R scripts at the appropriate times and communicates with them via standard input and standard output.
- Java API: Write the Map and Reduce operations in Java. The Java code, in turn, invokes `Runtime.exec()` or some equivalent to launch your R scripts

Call Example

Situation: In this example (ex4), the input data is several million lines of plain-text phone call records. Each CSV input line is of the format:
{ID}{date}, {caller num}, {caller carrier}, {dest num}, {dest carrier}, {length}

See: cdat2.csv

Run: cat cdat2.csv | ./mapper.R | sort | ./reducer.R

MSNBC Example

Run: less msnbc990928.seq | ./mapper.R | sort | ./reducer.R

Also, see: msnbc.Rmd

Rhipe Overview

```
https://www.datadr.org/index.html
```

```
https://www.datadr.org/install.html
```

A typical RHIPE call:

```
rhipe.job.def <- rhmr(
map= ... block of R code for Mapper
reduce= ... block of R code for Reducer
ifolder="/path/to/input" ,
ofolder="/path/to/output" ,
... a couple other RHIPE options
)
rhex( rhipe.job.ref )
```

My Research

- Bioinformatics: Random KNN (moderate n, large p)
- Molecular Dynamics: Approximate KNN (large n, small p for small molecules)
- Molecular Dynamics: Combine Approximate KNN with Random KNN? (large n, large p for proteins)
- Extensible Markov Models for Streaming Data