CS 686: Special Topics in Big Data

MapReduce

Lecture 17

Today's Schedule

- Assignment Schedule
- Deploying to Bass
- MapReduce
- MapReduce Examples

Today's Schedule

- **Assignment Schedule**
- Deploying to Bass
- MapReduce
- MapReduce Examples

Assignment Schedule (1/2)

- Design Document
- Project 1 Code Submission
- Paper Evaluation 4
- Project Retrospective

Assignment Schedule (2/2)

- This week, we'll have a lab on Wednesday
- Grading begins
  - I'll send out a sign-up sheet this evening

Today's Schedule

- Assignment Schedule
- **Deploying to Bass**
- MapReduce
- MapReduce Examples

Deploying to Bass

- Tip: use the pom.xml file provided in the **protobuf-example** repository:
  - https://github.com/cs686-bigdata/protobuf-example.git
- Most important: bundling protobuf jars if you're not using the version on bass
- Reminder: bass nodes have Java 7 installed

Today's Schedule

- Assignment Schedule
- Deploying to Bass
- **MapReduce**
- MapReduce Examples

Today's Subject: Google

- Google stores a **lot** of information
- Two big sources?
    1. Crawled web data (plus history)
    2. Log files
- These days Google is a lot more than just a search engine!
- MapReduce was designed for batch processing at **warehouse computing** scales

Crawling the Web

- The web is constantly changing!
- Can be represented as a graph:
  - Vertex: a web page
  - Edges: hyperlinks
- How do we make sense of all this?

Approaches

- Inverted indices
  - Normally, we map keys → values
  - With an inverted index, we map the values back to the keys
- **PageRank**
  - The basis of Google searches
  - Nowadays, Google's search algorithm is much more advanced

Logs

- Log files are almost as important as the main datasets we manage
- This is where we learn about what users are doing!
- *What is the most common Google search today?*
- *How many users logged into Facebook in Pheonix, AZ last week?*
- *What datacenter experiences the most power outages or hardware failures?*

MapReduce

- MapReduce is a **programming paradigm** to make developing parallel applications easy(ish?)
- Highly restrictive, but takes care of many details
  - No fault tolerance to worry about
  - No threads
- Consists of two phases:
  - Map
  - Reduce

<key, value> pairs

- The bread and butter of any MapReduce application are simple <key, value> pairs
  - Inputs
  - Outputs
- This is restrictive
  - Many times we have to rethink our problem in "MapReduce style"

Map

- Receives an input <k, v> pair
- Produces an intermediate <k, v> pair
- Intermediate pairs are grouped by the framework and then passed to the reducer

Reduce

- Receives intermediate <k, v-group> entries
- Merges the values together
- Output?
  - You guessed it! <k, v> pairs!

Data Types

- Everything is a byte array (aka String)
- Most MapReduce implementations offer some functionality to hide this fact from the developer
  - Automatically serializing numbers, for instance

Workflow: Popular URIs

- Map:
  - Read log file
  - Emit <URI, 1> pairs
- Reduce:
  - Add up the counts for each URI
  - Emit <URI, total> pairs

Sorting

- So we have a list of popular URIs sorted by… hmm, not sorted at all
- How can we sort?
  - One **bad** approach: having a single reducer
    - This is common for little jobs but not efficient
- Better approach: custom partitioners
  - Allows the developer to divide the keyspace across Reducers

Building an Inverted Index

- Map:
  - Read web page
  - Emit <word, URI>
- Reduce:
  - Sort by URI
  - Emit <word, list(URI 1, URI 2, URI 3, … URI N)>

Today's Schedule

- Assignment Schedule
- Deploying to Bass
- MapReduce
- **MapReduce Examples**

MapReduce Examples

- You can find several example MR applications on the bass nodes
- Run:
  - `hadoop jar /opt/cloudera/parcels/CDH-5.3.0-1.cdh5.3.0.p0.30/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar`
- For a full list

The “Hello World” of MapReduce

- Since printing “Hello World” on a bunch of machines isn’t all that impressive, we need something else
- One of the most common examples: Word Count
- This is a pleasingly parallel job:
  - Break our input file(s) up
  - Count the words on each line
  - Emit <word, #> pairs

Word Count

- With just this information, we can analyze a lot!
- What are the most common words in our languages?
- How does the frequency of words change over time?
- We can start updating our analysis to think about sentiment, spelling changes, and more
- Let's look at an example MapReduce application…