Course Notes, October 28, 2013 – Lecture 9

Eugene Shvarts
Stats 250 – Baines – Fall 2013 – UC Davis

Homework is due

Introducing code-swaps, and the procedure. Link is available at `sta250.github.io/Stuff/codeswaps` . Completing a code-swap report is due a week after the homework assignment due date.

Distributed File Systems

Today is more about computing, than about statistics.

When data gets big, it can't be read directly into R. So, distributed file systems (DFS). There needs to be a more sophisticated programming framework just to do basic data processing – you need a model which automatically abstracts out the process of finding the data and synchronizing it.

MapReduce

Two steps to programming for MapReduce: first you **map**, then you **reduce**.

- **Map**: For every data element, a function is applied to that element, and it returns a (key,value) pair.
- **Reduce**: For every element with the same key, a function is applied to combine the values. There could be potentially millions or billions of such pairs! You could take the sum of the values, or any useful statistic.

Classic example: Count all of the word instances in a string. One method is to treat each word in the string as its own 'computer', or 'data center' – each generates a key (corresponding to the word), and a value (for our current purposes, a 1 suffices). The shorthand here is `emit: (word,1)`. Next, for the reduce step, we combine all pairs with the same key, and then **reduce** in order to return a single value for each distinct key, each of which is a function of the values where the key is duplicated. Here, our reduce function is just a sum (and so, we simply count the words). 3 instances of 'Bob' produces 3 pairs (Bob, 1) (Bob, 1) (Bob, 1), which are reduced to (Bob, 3).

We can implement this in pseudocode; the slides provide the code. A note is that MapReduce works exclusively (usually?) in strings, and so the values must be parsed to integers to do the appropriate arithmetic. In this example, because strings can differentiate capitalization (A != a), 'angry' is a distinct key from 'Angry'.

Hadoop

Hadoop is written in Java, but the Hadoop Streaming API allows running using any executable or script. For this course, the most convenient method will probably be scripting in Python, and then proceeding through the Hadoop Streaming interface.

`Python implementation displayed in slides.` (The important detail here was the script used to enter the Python environment)

For MapReduce, the Reduce step is generally more tricky to piece together than the Map step. Many precautions are taken with regard to excess whitespace, new-line characters, and other possibly subtle formatting issues. Between the Map and Reduce step, the key-value pairs must be sorted according to the keys! This introduces some complexity, but it generally pales in comparison with what is required to keep track of all the counts at once in the Reduce step. Because the input consists of strings, dictionary sorting can be done by whatever our favorite ordering is.

Hadoop itself is an implementation of MapReduce. There are 4 pieces to deal with:

1. Namenode: master server or 'head node' which manages filesystem namespace and regulates access to files by clients.
2. Datanode: One per node in the cluster, which manages storage attached to the nodes that they run on. So far, very similar to how Gauss is structured.
3. Jobtracker: The headnode for tasks, rather than data. Manages the assignment of Map and Reduce tasks to the tasktrackers.
4. Tasktracker: Executes tasks, and handles data motion between Maps and Reduces.

Critical to Hadoop (rather, its typical use) is **fault tolerance**. The Hadoop File System (HDFS) has redundancy built into it. By default, each file is replicated three times (although this can be configured). Files are replicated across different datanodes. Large files are 'chunked' or 'sharded'. Default size is 64 MB, which aids transfer and replication. Uniform small chunk size aids for converting long run times on one machine into short run times utilizing many machines (think bag of little bootstraps, consensus Monte Carlo).

`Prof. Baines doing a code demo on screen.` (running Hadoop on just the one laptop; using framework but one datanode means no replication, etc.)

In order to run MapReduce using the code, the files need to be loaded into the HDFS first (once it has been loaded and is running). `hadoop fs` gives commands to the HDFS. So, `hadoop fs -ls` for instance will show you what's loaded into the HDFS directory. Then, hijinks / technical difficulties ensue. Okay, we're back.

`-copyFromLocal` from a local directory to some directory in the HDFS. Once the files are there, we need to run our routines. Need to use the Hadoop Streaming API, and tell Hadoop where the Map and Reduce scripts live. When using HSTREAMING, input directory needs to live on the HDFS, and the output directory must not exist yet. Hadoop produces a useful local HTML file which keeps track of the mapping and reducing with lots of diagonstic information and details. Note: all map steps must be completed before reducing can begin! Hence, optimizing map steps to avoid bottlenecks can improve performance.

Next, we need to copy our results off of HDFS to the local system; lo and behold, `-copyToLocal` does exactly what we think it might. Don't forget to clean up the HDFS afterwards; code is provided in slides.

Many algorithms that might benefit from MapReduce require **iteration**; fortunately Hadoop supports recursion, but utilizing the Streaming API makes direct chaining difficult. Other tools have been created that are more effective for doing more detailed statistical analysis – see the slides. **What next?** First, *debugging is horrible*. So, *abstraction is wonderful*. We'd rather work with C than Assembly, and R than C. Links to the following are in the slides.

- **Hive**: Project structures onto data, and utilize basic SQL-type queries. So, some database functionality. The part you want to not miss: We will use Hive in the next homework.
- **Pig**: Framework for easier data analysis using Hadoop. Consists of a higher-level SQL-style language called "Pig Latin". Essentially compiles Pig Latin programs into MapReduce tasks.
- **Mahout**: Scalable library for machine learning using Hadoop. Includes collaborative filtering, K-means, random forests, etc.

Hadoop is not on Gauss (oh no!). 'SLURM' commands which systems do what on Gauss; this doesn't mesh with Hadoop's controller for job management. Local install is fine for debugging; using just one slow node is relatively pointless though. In comes Amazon Web Services (AWS) to save the day – these are various tools for doing cloud computing, including a tool called ElasticMapReduce (EMR).

`Prof. Baines discusses 'Example 2' from slides for doing numerical computation.` In particular, from 'Example 3', for computing in-group variances we see a potential application for chaining together MapReduce instances. Once you Reduce to compute the means, take the same keys and change the value for elements x_{ij} of group i to $(x_{ij} - \overline{x_i})^2$ – then Reducing by summing and dividing by $n_i - 1$ produces the variance. Point was made of numerical instability that could be introduced by the repeated additions (but they can't alternate, as they're positive?).

`Prof. Baines doing a demo for the variance example on the screen.` Great! All worked; not much to say.

See you Wednesday!