Introduction to data science

Patrick Shafto

Department of Math and Computer Science

Plan for today

- Course reviews!
- Hadoop / Apache Spark!
- Homework!

Rate this class!

- Students can still respond until 11:59 pm (EST) on Friday, December 15 at
- https://sakai.rutgers.edu/portal/site/sirs
- Do it now!

Big data, what's the big deal?

- Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate to deal with them. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy. (https://en.wikipedia.org/wiki/Big_data)

- One simple example of this is: what happens if your data don't fit on a single machine?

Dealing with big data: mapreduce

- MapReduce is a framework for processing parallelizable problems across large datasets using a large number of computers (nodes)
    - cluster: if all nodes are on the same local network and use similar hardware)
    - grid: if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware
- Processing can occur on data stored either in a filesystem (unstructured) or in a database (structured).
- MapReduce can take advantage of the locality of data, processing it near the place it is stored in order to reduce the distance over which it must be transmitted.

Dealing with big data: mapreduce

- "Map" step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.

- "Shuffle" step: Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.

- "Reduce" step: Worker nodes now process each group of output data, per key, in parallel.

Dealing with big data: mapreduce

- MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source.

- Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative.

- While this process can often appear inefficient compared to algorithms that are more sequential (because multiple rather than one instance of the reduction process must be run), MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours.[12]

- The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

Dealing with big data: mapreduce

- A painfully simple example:
- Assume you have five files, and each file contains two columns (a key and a value in Hadoop terms) that represent a city and the corresponding temperature recorded in that city for the various measurement days.
- Of course we’ve made this example very simple so it’s easy to follow. You can imagine that a real application won’t be quite so simple, as it’s likely to contain millions or even billions of rows, and they might not be neatly formatted rows at all

Dealing with big data: mapreduce

- File 1:
    - Toronto, 20
    - Whitby, 25
    - New York, 22
    - Rome, 32
    - Toronto, 4
    - Rome, 33
    - New York, 18

Dealing with big data: mapreduce

- Out of all the data we have collected, we want to find the maximum temperature for each city across all of the data files (note that each file might have the same city represented multiple times).

- Using the MapReduce framework, we can break this down into five map tasks, where each mapper works on one of the five files and the mapper task goes through the data and returns the maximum temperature for each city.

- For example, the results produced from one mapper task for the data above would look like this:

    - (Toronto, 20) (Whitby, 25) (New York, 22) (Rome, 33)

Dealing with big data: mapreduce

- Let's assume the other four mapper tasks (working on the other four files not shown here) produced the following intermediate results:

- (Toronto, 18) (Whitby, 27) (New York, 32) (Rome, 37)(Toronto, 32) (Whitby, 20) (New York, 33) (Rome, 38)(Toronto, 22) (Whitby, 19) (New York, 20) (Rome, 31)(Toronto, 31) (Whitby, 22) (New York, 19) (Rome, 30)

Dealing with big data: mapreduce

- All five of these output streams would be fed into the reduce tasks (with an optimized shuffle to distribute the remaining task efficiently!), which combine the input results and output a single value for each city, producing a final result set as follows:
- (Toronto, 32) (Whitby, 27) (New York, 33) (Rome, 38)

Mapreduce / hadoop limitations

- Mapreduce is a core component of Hadoop
- Mapreduce (and hadoop) are optimized for out of memory computation. This means that if one has to recompute often, one must repeat the (slow) step of reading and writing from disk.
- This is a big limitation for machine learning where:
    - computations are often iterative
    - data is often streamed
- (not the only limitation)

MapReduce vs Spark

- MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk.

- Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

    - RDD = Resilient distributed dataset, a read-only multiset of data items distributed over a cluster of machines

    - This allows multiple queries of the data without reading or writing to disk

MapReduce vs Spark

- MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk.

- Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

    - RDD = Resilient distributed dataset, a read-only multiset of data items distributed over a cluster of machines

    - This allows multiple queries of the data without reading or writing to disk

Apache spark in jupyter netbooks

- http://spark.apache.org/
- Quick start:
    - http://spark.apache.org/docs/latest/quick-start.html
- Examples
    - http://spark.apache.org/examples.html
    - https://github.com/jadianes/spark-py-notebooks
- Installing:
    - http://www.cloudera.com/documentation/enterprise/5-5-x/topics/spark_ipython.html

Homework for Monday

- Make sure demo of tensor flow is running on your computer
- Read the alpha go paper (Silver et al, 2016). It is on blackboard.