Lecture 34: Distributed Computing: Data

- The Unix operating system provides a simple model for accessing data.
- The basic abstraction is a *stream* of bytes (which most often translates to a stream of text).
- At the programmer level, have simple operations: *read* and *write* sequences of bytes.
- On top of this is built the reading of clusters of bytes (numerals, lines, words separated by whitespace, etc.)
- Programs typically have a standard input, standard output, and standard error: three streams.
- Hence, programs can be fitted together into series of programs—*pipelines*:

> "We should have some ways of coupling programs like [a]
> garden hose—screw in another segment when it becomes
> necessary to massage data in another way."
> –Douglas McIlroy [1964].

Pipeline Example

- Example (from Lecture 9): Twenty most common words in a manuscript:

```
tr -c -s ’[:alpha:]’ ’[\n*]’ < FILE | \
sort | \
uniq -c | \
sort -n -r -k 1,1 | \
sed 20q
```

- This example shows only limited opportunities for parallelism (sorting is a bottleneck).
- In general there may be more, but generally limited to number of segments in the pipeline.

Big Data

- So what happens with huge amounts of data?
- Examples (from Anthony Joseph, although a bit dated):
  - Facebook's daily logs: 60 Terabytes (60,000 Gigabytes)
  - 1,000 genomes project: 200 Terabytes
  - Google web index: 10+ Petabytes (10,000,000 Gigabytes)
  - Disk speeds: order of 100MBytes/sec, maybe a few times that for SSDs (solid-state disks).
- Bottom line: need to break up—*distribute*—data and attach computation to each chunk.

Apache Spark

- The Apache Spark framework (first developed here) attempts to provide a simple view of large datasets, permitting intuitive and flexible constructions analogous to Unix pipelines.
- A *Resilient Distributed Dataset (RDD)* is a collection of values or *key-value pairs* (as in a Python dictionary).
- Provides a variety of familiar operations on them:
  - Unix-like: sort, count, distinct (uniq), pipe.
  - SQL-like: union, intersection, join (multiple tables).
  - General sequence operations: map, filter, reduce (accumulate).
- Provides for partitioning of data across machines, allowing operations to be implemented in parallel.
- Adds fault tolerance, load redistribution, and monitoring.

Map-Reduce

- One type of such distributed computation, developed at Google, was generically called *map-reduce*.
- Google has since moved on to other techniques, but map-reduce is still interesting, and suitable for implementation on the Spark framework.
- Basic idea of a map-reduce application:
  - *Map* data into key-value pairs. This operation is distributed over many segments of data and processors, each of results in a set of such pairs.
  - *Shuffle* the pairs so that those with matching keys are grouped together.
  - *Reduce* (accumulate) the values for each key, producing as a final product a mapping from keys to collected values. Each individual reduction can proceed independently of others.
- By plugging in appropriate mapping and reduction functions, can get a large variety of overall functions.