CMSC 714
Lecture 15
Cloud Computing - MapReduce

Alan Sussman

Notes

- CUDA project grades posted
  - Email Ryan if you have questions about your grade
- Research proposals under evaluation
  - Feedback within a couple days
- Midterm exam on April 15

MapReduce

- Both a programming model and a Google implementation for processing large data sets on clusters of commodity computers w/o a fast network
  - targeted data is mainly Web documents and related data, but has been applied to (many) other domains
- Programming model is functional, and goes back to Lisp (in 1960's!)

MapReduce (cont.)

- Functional programming model, so processing order does not matter – user writes 2 functions:
  - **Map** takes an input (key, value) pair and produces a set of intermediate (key, value) pairs
  - **Reduce** takes a key, and all the corresponding values for the key from the intermediate pairs, and merges the values into a new set of values (sometimes just 1 value)
    - the intermediate values are given to the function via an iterator (helps when all values for a key don't fit into memory)
- Main input/output data type is strings, but can work on any type internally
  - Does require type conversion, which can become expensive

MapReduce (cont.)

- Implementation - runtime system does the parallelization onto the cluster
  - master/worker model – 1 master assigns map and reduce tasks to available worker machines
  - relies heavily on GFS – Google distributed file system
  - partition input data – called *splits*
  - schedule execution across cluster – try to have map tasks assigned near (in network terms) where the input data is located, and similarly have reduce tasks assigned near where map task outputs are written
  - deal with machine failures – restart failed tasks on other worker machines, and ensure each task only outputs once
    - if master fails, restart from checkpoint
  - manage communication between machines
- Several refinements/optimizations to give users more control over execution if desired, to provide additional functionality, to improve performance in some cases, to help with debugging, etc.

MapReduce more recently

- Lots of work over last > 20 years on open source implementations (e.g., Hadoop)
  - With many optimizations to improve performance, ease programmability
  - And for reliability – not true that a single master task for a job is adequate
- Relying on distributed file system for storing input and output (and some intermediate results) can be problematic for many applications
  - Locality is important for performance

MapReduce vs. Parallel DBMSs

- A response from the relational DB community to the popularity and claims of MapReduce advocates
  - a shortened version of a SIGMOD 2009 conference paper for a more general audience
- Overall claim is that MR is complementary to pDBMSs, not a replacement
- Advantages of MR include:
  - Extract-Transform-Load applications, including loading data into a DBMS
  - Complex analytics – data mining, data clustering
  - Semi-structured data – no schema, but (key,value) pairs
  - Easy software install, for “quick and dirty analyses”
  - Cost – Hadoop is open source, but no open source pDBMSs
  - MR is a powerful tool for some applications

MR vs. pDBMSs (cont.)

- Advantages of pDBMSs include:
  - Performance, even on tasks that appear well-suited to MR
    - results in paper mitigated by comparing solid commercial pDBMSs against Hadoop, a relatively new open source implementation (at the time)
  - Data parsed when loaded into DBMS, so not parsed again when executing queries
  - Performance gains from compressing data
    - and hard to get those gains with semi-structured MR data in a distributed file system
  - Pipelined execution of compiled SQL operations from streaming of data between operators, instead of writing intermediate data into distributed file system for MR
  - Static query planning vs. MR runtime work scheduling
    - but MR can adapt better to heterogeneous hardware

MR vs. pDBMSs (cont.)

- This is the beginning of the DB research community starting to take noSQL DBs seriously
  - Leading to a proliferation of heavily researched and eventually used DBs beyond relational
    - Mainly for various types of *semi-structured* data, so no relational DB schema defined
    - E.g., column stores, key/value stores, document stores, graph DBs, data streaming systems, …