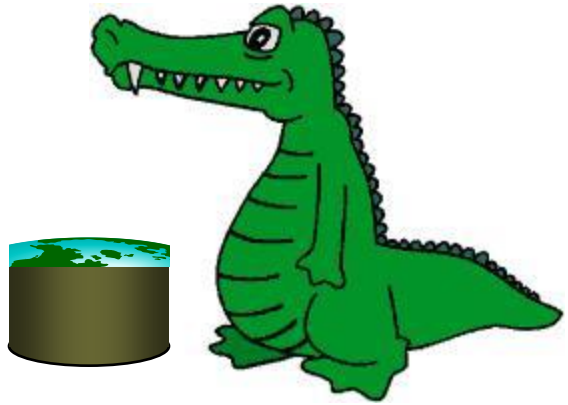
CAP4770/5771
Introduction to Data Science
Fall 2016

University of Florida, CISE Department
Prof. Daisy Zhe Wang



Based on notes from CS194 at UC Berkeley by Michael Franklin, John Canny, and Jeff Hammerbacher

Logistics

- Lab 4 finished, Lecture 6 this week
- Lab 5 material will be released this Friday, due Thursday 11:59pm
  - JAVA + AWS/EMR
- Lecture 7 next Monday
- Lab 5 in class next Wed.
  - Extra prep: AWS, EMR setup and tutorial
- NIST DSE Introduction + QA next Friday
- No office hour next Wed.

Review

- Machine Learning
  - Supervised
  - Unsupervised
- Supervised Learning
  - Classification and Regression
  - K-NN
  - Naïve Bayes
  - Linear/Logistic Regression
  - SVM

Supervised Learning Pipeline

1. Application Data: D
2. Training Data Annotation/Labeling: D→y_T
3. Feature Extraction D → X_T
4. Model Selection: KNN, NB, LR, LogR, SVM
5. Training: Model Fitting X_T, y_T → f(X)
6. Testing: Inference/Prediction X_t→y_t-hat
7. Evaluation: e.g., precision/recall
8. Analysis and Iterate: e.g., more labeling, features…

Main techniques

- Feature extraction from different data types, e.g., structured, logs, text, image
- Learning
  - Model: KNN, NB, LR, LogR, SVM
  - Model parameters, error function: SSE/ML, optimization: minimization/maximization
- Inference
  - For each of KNN, NB, LR, LogR, SVM
  - How to use learned model f(X) to predict new numerical or categorical values y_t-hat
- Evaluation: P-R, Type I/II error

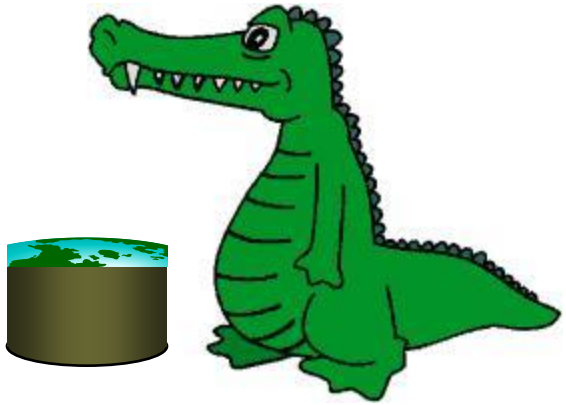
Map/Reduce: Simplified Data Processing on Large Clusters

Parallel/Distributed Computing

Distributed File System

M/R Programming Model

Parallel Analytics using M/R



Adapted Slides from Jeff Ullman, Anand Rajaraman and Jure Leskovec from Stanford

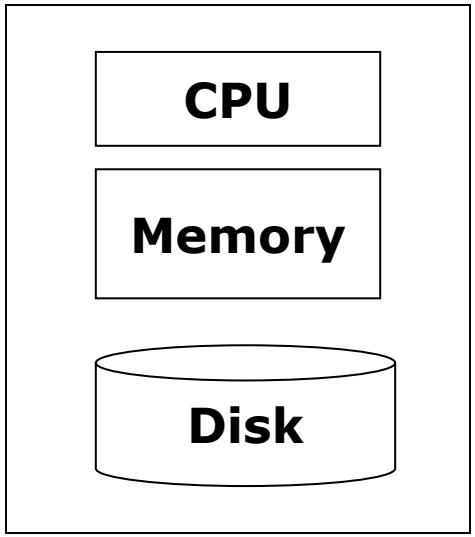
Parallel Computing

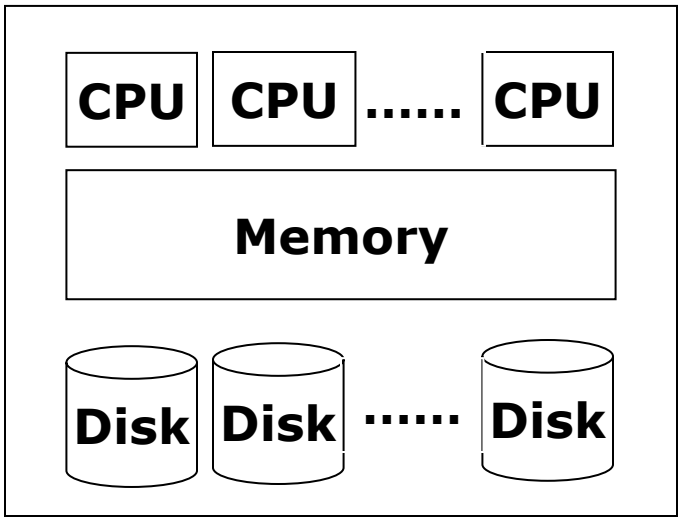
- MapReduce is designed for parallel computing
- Before MapReduce
  - Enterprise: a few super-computers, parallelism is achieved by parallel DBs (e.g., Teradata)
  - Science, HPC: MPI, openMP over commodity computer clusters or over super-computers for parallel computing, do not handle failures (i.e., Fault tolerance)
- Today:
  - Enterprise: map-reduce, parallel DBs
  - Science, HPC: openMP, MPI and map-reduce

MapReduce can apply over both multicore and **distributed environment**

Data Analysis on Single Server

- Many data sets can be very large
  - Tens to hundreds of terabytes
- Cannot always perform data analysis on large datasets on a single server (why?)
  - Run time (i.e., CPU), memory, disk space





Motivation: Google's Problem

- 20+ billion web pages * 20KB = 400+ TB
  - ~1000 hard drives to store the web
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web

- Takes even more to do something useful with the data!
  - CPU
  - Memory

Commodity Clusters

- Recently standard commodity architecture for such problems:
  - Cluster of commodity Linux nodes
  - Gigabit ethernet interconnect

- How to organize computations on this architecture?

Distributed Computing

- Challenges:
  - Distributed/parallel programming is hard
    - How to distribute Data? Computation? Scheduling? Fault Tolerant? Debugging?
  - On the software side: what is the programming model?
  - On the hardware side: how to deal with hardware failures?
- MapReduce/Hadoop addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

Stable storage

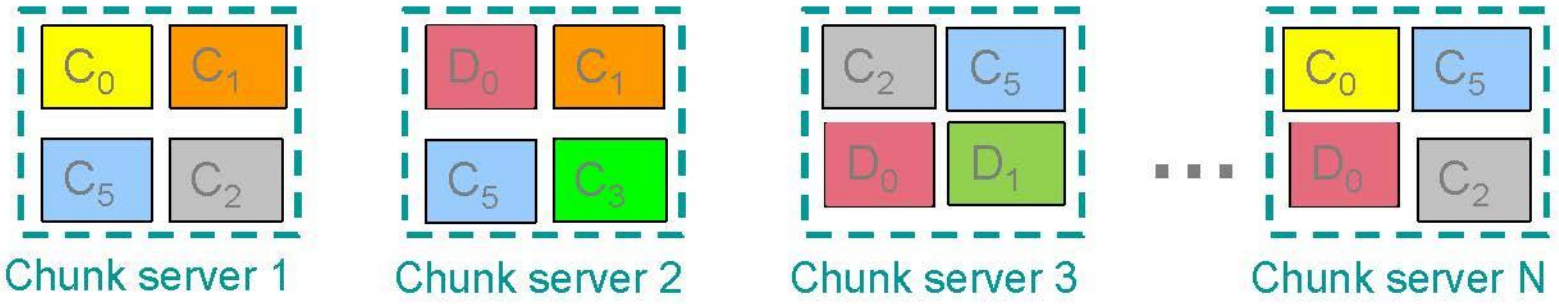
- First order problem: if nodes can fail, how can we store data persistently?
- Idea:
  - Store files multiple times for reliability
- Solution: Distributed File System
  - Provides global file namespace
  - Google GFS; Hadoop HDFS
- Typical Usage Pattern
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

- Chunk Servers
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- Master node
  - a.k.a. Name Nodes in HDFS
  - Stores metadata
  - Likely to be replicated as well
- Client library for file access
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

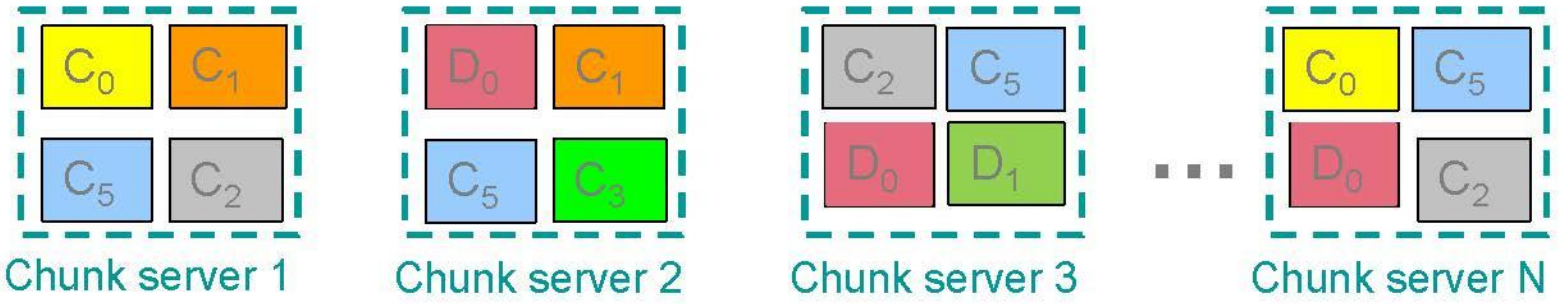
Reliable Distributed File System

- Data kept in "chunks" spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



Reliable Distributed File System

- Data kept in "chunks" spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



Bring computation directly to the data!

Chunk servers also serve as compute servers

Simpole Programming Model for Distributed Computing: MapReduce

- Large-Scale Data Processing
  - Want to use 1000s of CPUs, But don't want the hassle of scheduling, synchronization, etc.
- MapReduce provides
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling and synchronization
  - Monitoring & status updates

MapReduce Basics

- MapReduce
  - Programming model similar to Lisp
  - (and other functional languages)

- General pattern for parallel/distributed computing: Many problems can be phrased using Map and Reduce functions

- Simple and intuitive to implement an algorithm in MapReduce API

Example Problem: Word Count

- We have a huge text file: doc.txt or a large corpus of documents: docs/*
- Count the number of times each distinct word appears in the file..
  - Apps?
- Sample application: analyze web server logs to find popular URLs

Word Count (2)

- Case 1: File too large for memory, but all <word, count> pairs fit in memory
- Case 2: File on disk, too many distinct words to fit in memory
  - Count occurrences of words

```
words(doc.txt) | sort | uniq -c
```

  - `words` takes a file and output the words in it, one word a line
  - Naturally capture the essence of MapReduce and naturally parallelizable

3 steps of MapReduce

- Sequentially read a lot of data
- **Map:** Extract something you care about
- **Group by key:** Sort and shuffle
- **Reduce:** Aggregate, summarize, filter or transform
- Output the result

For different problems, steps stay the same,
Map and Reduce change to fit the problem

MapReduce: more detail

- Input: a set of key/value pairs
- Programmer specifies two methods
  - Map(k,v) → list(k1,v1)
    - Takes a key-value pair and outputs a set of key-value pairs (e.g., key is the file name, value is a single line in the file)
    - One map call for every (k,v) pair
  - Group by key is executed by default: Sort(list(k1,v1)) → list (k1, list(v1))
  - Reduce(k1, list(v1)) → list (k1, v2)
    - All values v1 with the same key k1 are reduced together to produce a single value v2
    - There is one reduce call per unique key k1

InputFormat → <K,V> pairs

- InputFormat: transform on-disk data representation on HDFS to in-memory key-value

- Example: InputFormat for plain text files. Files are broken into lines. Either linefeed or carriage-return are used to signal end of line.

  - TextInputFormat: Keys are the position in the file, and values are the line of text.

  - KeyValueTextInputFormat: Each line is divided into key and value parts by a separator byte. If no such a byte exists, the key will be the entire line and value will be empty.