CIS 833 – Information Retrieval and Text Mining

Lecture 2

# Introduction to MapReduce

August 27, 2015

Credits for slides: Hofmann, Mihalcea, Mobasher, Mooney, Schutze.

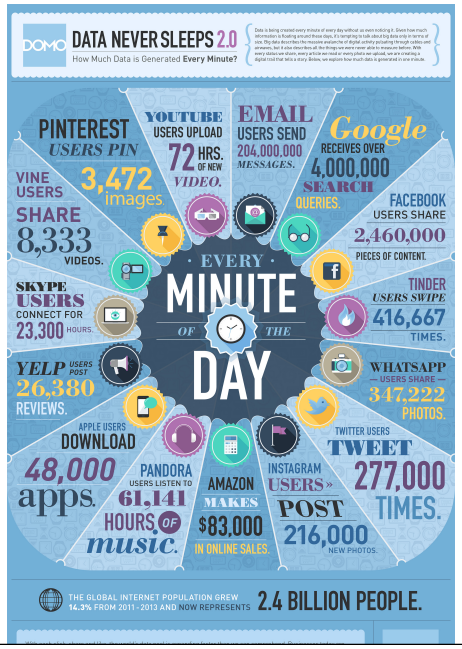
## Required Reading

- MapReduce & Hadoop
  - **Data-Intensive Text Processing with MapReduce** (Lin and Dyer) http://www.umiacs.umd.edu/~jimmylin/book.html
    - Chapter 1, Introduction
- Next time:
  - Chapter 2, MapReduce
  - **MapReduce: Simplified Data Processing on Large Clusters** by Jeffrey Dean and Sanjay Ghemawat from Google Labs

# What is MapReduce?

- Programming model for expressing distributed computations at a massive scale
- Execution framework for organizing and performing such computations
- Open-source implementation called Hadoop
- Pig provides a higher level language, Pig Latin.


## Data Explosion in 2014 Minute by Minute - Infographic




http://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic/

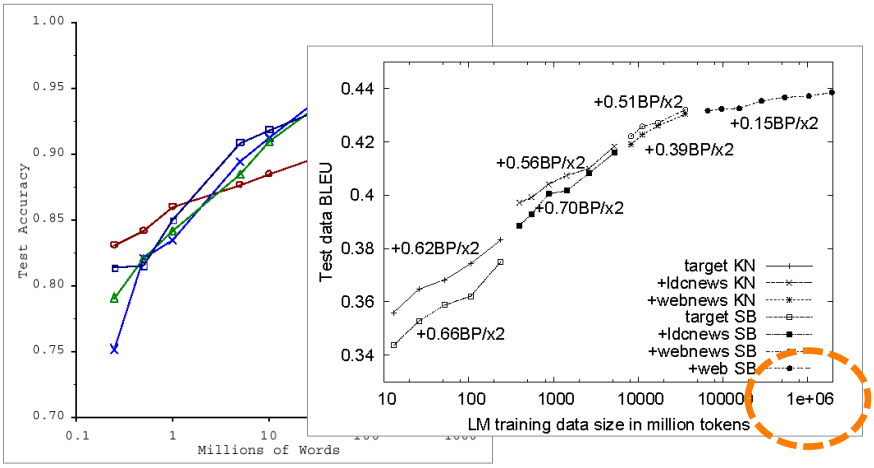
# How much data?

- Google processes 100 PB/day; 3 million servers
- Facebook has 300 PB + 600 TB/day; 35% of world's photos
- YouTube 1000 PB video storage; 4 billion views/day
- Twitter processes 100TB/day; 124 billion tweets/year
- SMS messages – 6.1T per year
- US Cell Calls – 2.2T minutes per year
- US Credit cards - 1.4B Cards; 20B transactions/year

https://followthedata.wordpress.com/2014/06/24/data-size-estimates/#comments

http://www.slideshare.net/kmstechnology/big-data-overview-2013-2014

# No data like more data!




How do we get here if we're not Google?

(Banko and Brill, ACL 2001) [**Scaling to Very Very Large Corpora for Natural Language Disambiguation**]

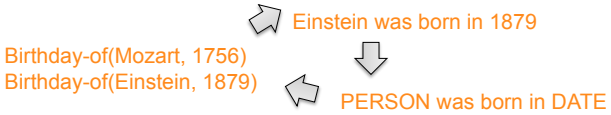
(Brants et al., EMNLP 2007) [**Large Language Models in Machine Translation**]

# What to do with more data?

For example:

- Answer factoid questions
  - Pattern matching on the Web
  - Works amazingly well

    **Who shot Abraham Lincoln? → X shot Abraham Lincoln**
- Learn relations
  - Start with seed instances
  - Search for patterns on the Web
  - Use patterns to find more instances




(Brill et al., TREC 2001; Lin, ACM TOIS 2007)
(Agichtein and Gravano, DL 2000; Ravichandran and Hovy, ACL 2002; … )
(Carlson et al., WSDM 2010) – *Coupling semi-supervised learning for information extraction*

# Large scale data processing

- Want to process lots of data
- Want to parallelize across hundreds/thousands of CPUs
- … Want to make this easy

Cloud computing seems to provide a solution!

# What is cloud computing?

- Cloud computing means many different things:
  - Large-data processing
  - Rebranding of web 2.0
  - Utility computing
  - Everything as a service

# Rebranding of web 2.0

- Rich, interactive web applications
  - Clouds refer to the servers that run them
  - AJAX as the de facto standard
  - Examples: Facebook, YouTube, Gmail, …
- "The network is the computer"
  - User data is stored "in the clouds"
  - Rise of the netbook, smartphones, etc.
  - Browser *is* the OS

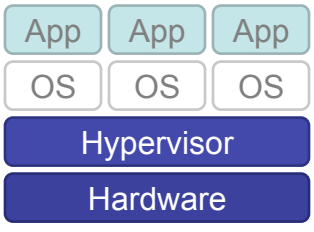
http://www.infoworld.com/article/2609165/web-browsers/10-reasons-the-browser-is-becoming-the-universal-os.html
http://www.theguardian.com/technology/2014/apr/09/software-internet

# Utility Computing

- What?
  - Computing resources as a metered service ("pay as you go")
  - Ability to dynamically provision virtual machines
- Why?
  - Cost: capital vs. operating expenses
  - Scalability: "infinite" capacity
  - Elasticity: scale up or down on demand
- Does it make sense?
  - Benefits to cloud users
  - Business case for cloud providers

# Enabling Technology: Virtualization


Traditional Stack




Virtualized Stack

A hypervisor, also called virtual machine monitor (VMM), allows multiple operating systems to run concurrently on a host computer.

## Everything as a Service

- Utility computing = Infrastructure as a Service (IaaS)
  - Why buy machines when you can rent cycles?
  - Examples: Amazon's Elastic Compute Cloud (EC2), Rackspace, IBM SmartCloud
- Platform as a Service (PaaS)
  - Give me nice API and take care of the maintenance, upgrades, …
  - Example: Google App Engine
- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail, Salesforce

## Who cares?

- Ready-made large-data problems
  - Lots of user-generated content
  - Even more user behavior data
  - Examples: Facebook friend suggestions, Google ad placement
  - Business intelligence: gather everything in a data warehouse and run analytics to generate insight
- Utility computing
  - Provision Hadoop clusters on-demand in the cloud
  - Lower barrier to entry for tackling large-data problem
  - Commoditization and democratization of large-data capabilities

## Cloud Resources

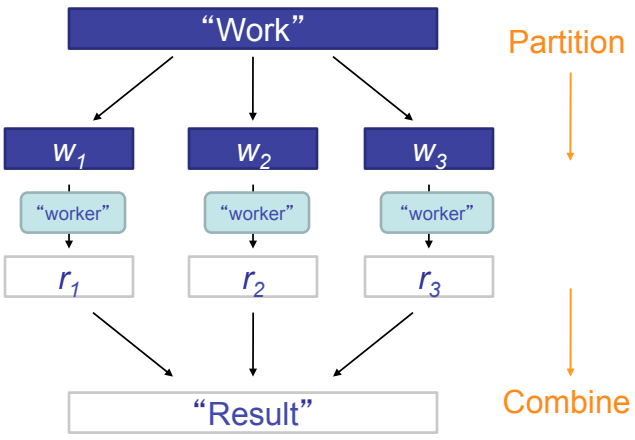
- Hadoop on your local machine
- Hadoop in a virtual machine on your local machine
- Hadoop on the Beocat - Cloudera Hadoop cluster
- Hadoop in the clouds with Amazon EC2
- Hadoop on the Google/IBM cluster
- …

## Hadoop "modes"

- Standalone (or local) mode
  - No daemons running, everything runs in a single JVM.
  - There is no HDFS! Uses local files.
  - Suitable for running MapReduce programs during development – makes it easy to debug and test them.
- Pseudo-distributed (or single-node) mode
  - Hadoop daemons run on the local machine, thus simulating a cluster on a small scale.
- Fully distributed mode
  - Hadoop daemons run on a cluster of machines.

## How do we scale up?

## Divide and Conquer



## Parallelization Challenges

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?

What is the common theme of all of these problems?

## Common Theme?

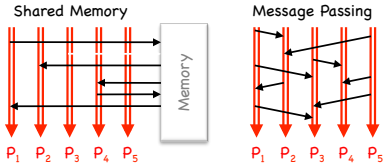
- Parallelization problems arise from:
  - Communication between workers (e.g., to exchange state)
  - Access to shared resources (e.g., data)
- Thus, we need a synchronization mechanism

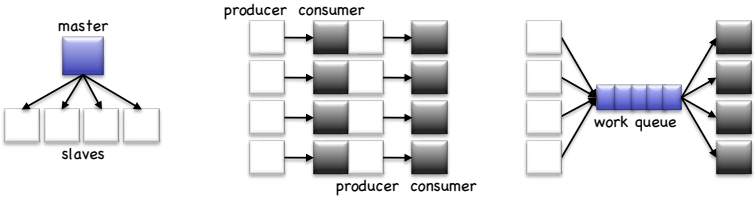
# Managing Multiple Workers

- Difficult because
  - We don't know the order in which workers run
  - We don't know when workers interrupt each other
  - We don't know the order in which workers access shared data
- Thus, we need:
  - Semaphores (lock, unlock)
  - Conditional variables (wait, notify, broadcast)
  - Barriers
- Still, lots of problems:
  - Deadlock, livelock, race conditions...
  - Dining philosophers, sleeping barbers, cigarette smokers...
- Moral of the story: be careful!

# Current Tools

- Programming models
  - Shared memory (pthreads)
  - Message passing (MPI)
- Design Patterns
  - Master-slaves
  - Producer-consumer flows
  - Shared work queues



## Current Tools Drawbacks

- Concurrency is difficult to reason about
- Concurrency is even more difficult to reason about
  - At the scale of datacenters (even across datacenters)
  - In the presence of failures
  - In terms of multiple interacting services
- Not to mention debugging…
- The reality:
  - Lots of one-off solutions, custom code
  - Write you own dedicated library, then program with it
  - Burden on the programmer to explicitly manage everything

## What's the point of MapReduce?

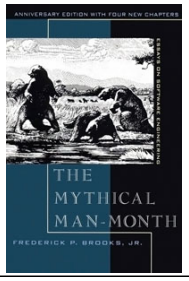
- It's all about the right level of abstraction
  - The von Neumann architecture has served us well, but is no longer appropriate for the multi-core/cluster environment
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
- Separating the *what* from *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework ("runtime") handles actual execution

The datacenter *is* the computer!

# "Big Ideas"

- Scale "out", not "up"
  - Limits of symmetric multi-processing (SMP) machines and large shared-memory machines
- Move processing to the data
  - Clusters have limited bandwidth
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable
- Seamless scalability
  - From the mythical man-month to the tradeable machine-hour

"adding manpower to a late software project makes it later"



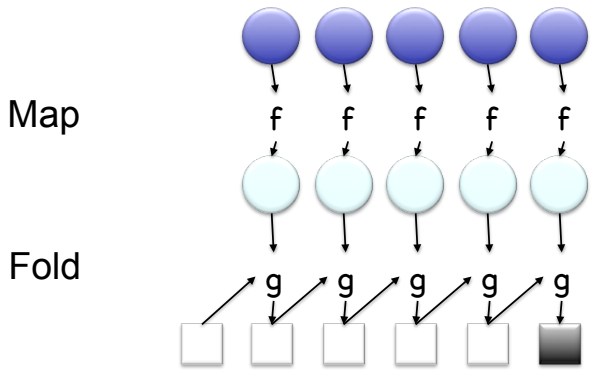

# Typical Large-Data Problem

- Iterate over a large number of records *Map*
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results *Reduce*
- Generate final output

Key idea: provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)

## Roots in Functional Programming




## Warm up: Word Count

- We have a large file of words, one word to a line
- Count the number of times each distinct word appears in the file
- Sample application: analyze web server logs to find popular URLs

## Word Count (2)

- Case 1: Entire file fits in memory
- Case 2: File too large for memory, but all <word, count> pairs fit in memory
- Case 3: File on disk, too many distinct words to fit in memory
  - `sort datafile | uniq -c`

## Word Count (3)

- To make it slightly harder, suppose we have a large corpus of documents
- Count the number of times each distinct word occurs in the corpus
  - `words(docs/*) | sort | uniq -c`

  where `words` takes a file and outputs the words in it, one to a line
- The above captures the essence of MapReduce
  - Great thing is it is naturally parallelizable