CS162
Operating Systems and Systems Programming
Lecture 24

Capstone: Cloud Computing

December 2, 2013

Anthony D. Joseph and John Canny

http://inst.eecs.berkeley.edu/~cs162

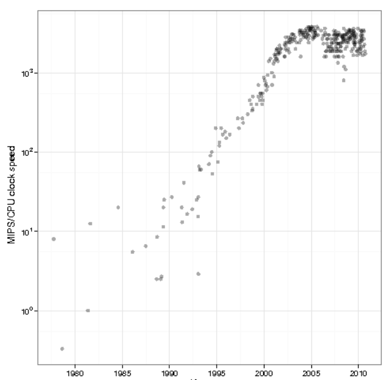
Goals for Today

- Distributed systems
- Cloud Computing programming paradigms
- Cloud Computing OS

Note: Some slides and/or pictures in the following are adapted from slides Ali Ghodsi.

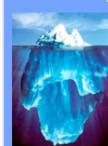
Background of Cloud Computing

- 1990: Heyday of parallel computing, multi-processors
  - 52% growth in performance per year!
- 2002: The thermal wall
  - Speed (frequency) peaks, but transistors keep shrinking
- The Multicore revolution
  - 15-20 years later than predicted, we have hit the performance wall



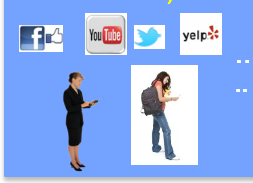
Sources Driving Big Data

It's All Happening On-line

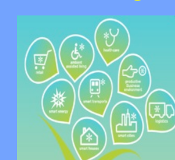


Every:
Click
Ad impression
Billing event
Fast Forward, pause,..
Friend Request
Transaction
Network message
Fault
…

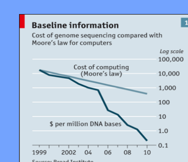
User Generated (Web & Mobile)



Internet of Things / M2M

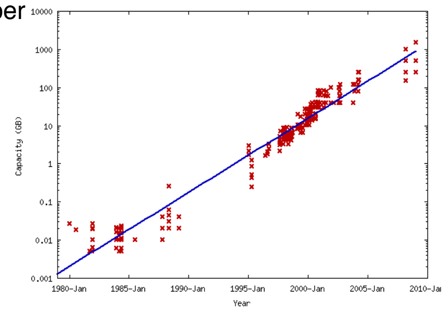


Scientific Computing

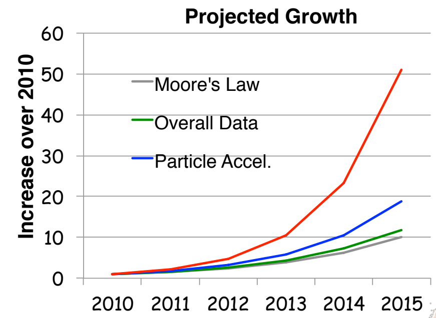


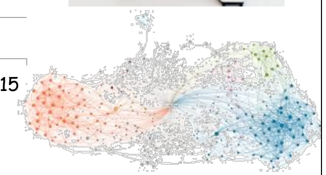
Data Deluge

- Billions of users connected through the net
  - WWW, FB, twitter, cell phones, …
  - 80% of the data on FB was produced last year
- Storage getting cheaper
  - Store more data!



Data Grows Faster than Moore's Law



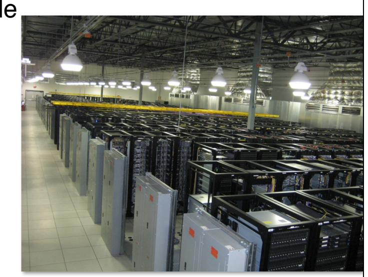


At the same time…

- Amount of stored data is exploding…

Solving the Impedance Mismatch

- Computers not getting faster, and we are drowning in data
  - How to resolve the dilemma?
- Solution adopted by web-scale companies
  - Go massively *distributed* and *parallel*



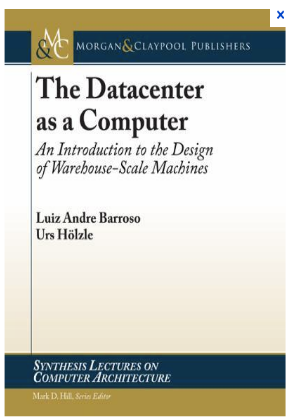
Enter the World of Distributed Systems

- Distributed Systems/Computing
  - *Loosely coupled* set of computers, communicating through message passing, solving a common goal
- Distributed computing is *challenging*
  - Dealing with *partial failures* (examples?)
  - Dealing with *asynchrony* (examples?)
- Distributed Computing versus Parallel Computing?
  - distributed computing=parallel computing + partial failures

Dealing with Distribution

- We have seen several of the tools that help with distributed programming
  - Message Passing Interface (MPI)
  - Distributed Shared Memory (DSM)
  - Remote Procedure Calls (RPC)
- But, distributed programming is still very hard
  - Programming for scale, fault-tolerance, consistency, …

The Datacenter is the new Computer



- *"Program"* == Web search, email, map/GIS, …
- *"Computer"* == 10,000's computers, storage, network
- Warehouse-sized facilities and workloads
- *Built from less reliable components than traditional datacenters*

Datacenter/Cloud Computing OS

- If the datacenter/cloud is the new computer
  - What is its **Operating System**?
  - Note that we are not talking about a host OS

Classical Operating Systems

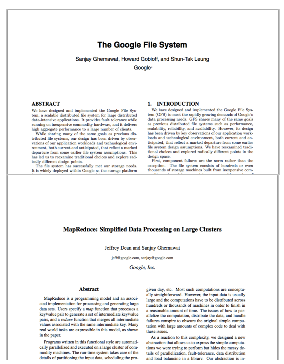
- Data sharing
  - Inter-Process Communication, RPC, files, pipes, …
- Programming Abstractions
  - Libraries (libc), system calls, …
- Multiplexing of resources
  - Scheduling, virtual memory, file allocation/protection, …

Datacenter/Cloud Operating System

- Data sharing
  - Google File System, key/value stores
- Programming Abstractions
  - Google MapReduce, PIG, Hive, Spark
- Multiplexing of resources
  - Apache projects: Mesos, YARN (MRv2), ZooKeeper, BookKeeper, …

Google Cloud Infrastructure

- Google File System (GFS), 2003
  - Distributed File System for entire cluster
  - Single namespace
- Google MapReduce (MR), 2004
  - Runs queries/jobs on data
  - Manages work distribution & fault-tolerance
  - Colocated with file system


The Google File System
Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google

MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean and Sanjay Ghemawat
Google, Inc.

- Apache open source versions Hadoop DFS and Hadoop MR

GFS/HDFS Insights

- *Petabyte* storage
  - Files split into large blocks (128 MB) and replicated across several nodes
  - Big blocks allow high throughput sequential reads/writes
- Data *striped* on hundreds/thousands of servers
  - Scan 100 TB on 1 node @ 50 MB/s = 24 days
  - Scan on 1000-node cluster = 35 minutes

GFS/HDFS Insights (2)

- *Failures* will be the norm
  - Mean time between failures for 1 node = 3 years
  - Mean time between failures for 1000 nodes = 1 day
- Use *commodity* hardware
  - Failures are the norm anyway, buy cheaper hardware
- No complicated consistency models
  - Single writer, append-only data

MapReduce Insights

- Restricted key-value model
  - Same **fine-grained operation** (Map & Reduce) repeated on big data
  - Operations must be **deterministic**
  - Operations must be **idempotent/no side effects**
  - Only communication is through the shuffle
  - Operation (Map & Reduce) output saved (on disk)

What is MapReduce Used For?

- At **Google**:
  - Index building for Google Search
  - Article clustering for Google News
  - Statistical machine translation
- At **Yahoo!**:
  - Index building for Yahoo! Search
  - Spam detection for Yahoo! Mail
- At **Facebook**:
  - Data mining
  - Ad optimization
  - Spam detection

MapReduce Pros

- Distribution is completely **transparent**
  - Not a single line of distributed programming (ease, correctness)
- Automatic **fault-tolerance**
  - Determinism enables running failed tasks somewhere else again
  - Saved intermediate data enables just re-running failed reducers
- Automatic **scaling**
  - As operations as side-effect free, they can be distributed to any number of machines dynamically
- Automatic **load-balancing**
  - Move tasks and speculatively execute duplicate copies of slow tasks (*stragglers)*

MapReduce Cons

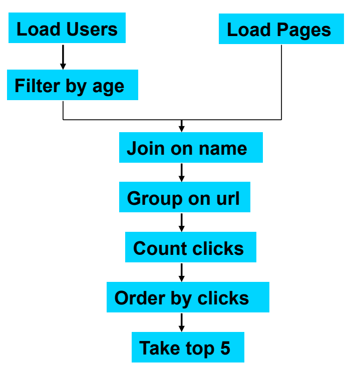
- Restricted programming model
  - Not always natural to express problems in this model
  - Low-level coding necessary
  - Little support for iterative jobs (lots of disk access)
  - High-latency (batch processing)
- Addressed by follow-up research
  - **Pig** and **Hive** for high-level coding
  - **Spark** for iterative and low-latency jobs

Pig

- High-level language:
  - Expresses sequences of MapReduce jobs
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc)
  - Easy to plug in Java functions
- Started at Yahoo! Research
  - Runs about 50% of Yahoo!'s jobs

Example Problem

Given *user data* in one file, and *website data* in another, find the *top 5 most visited pages by users aged 18-25*



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

In MapReduce

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

In Pig Latin

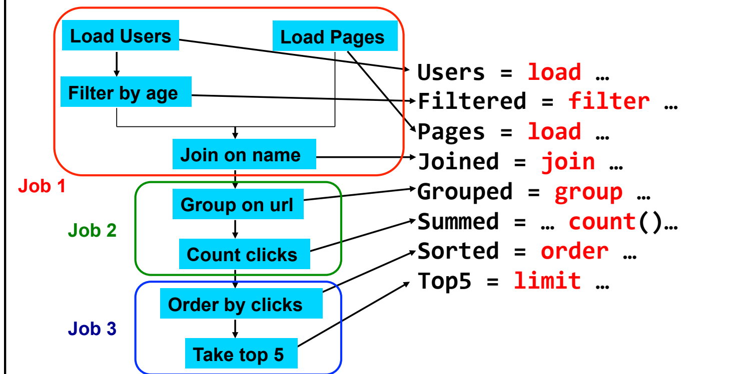
```
Users    = load 'users' as (name, age);
Filtered = filter Users by
                 age >= 18 and age <= 25;
Pages    = load 'pages' as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                  count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into 'top5sites';
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

Translation to MapReduce

Notice how naturally the components of the job translate into Pig Latin.



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope0

Hive

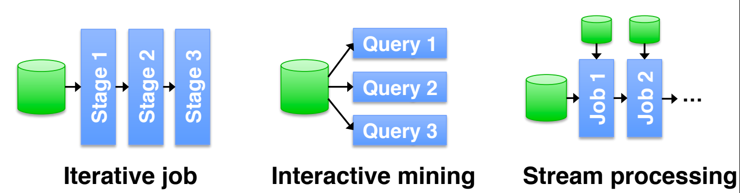
- Relational database built on Hadoop
  - Maintains table schemas
  - SQL-like query language (which can also call Hadoop Streaming scripts)
  - Supports table partitioning, complex data types, sampling, some query optimization
- Developed at Facebook
  - Used for many Facebook jobs

Spark Motivation

Spark Lightning-Fast Cluster Computing

Complex jobs, interactive queries and online processing all need one thing that MR lacks:

Efficient primitives for **data sharing**



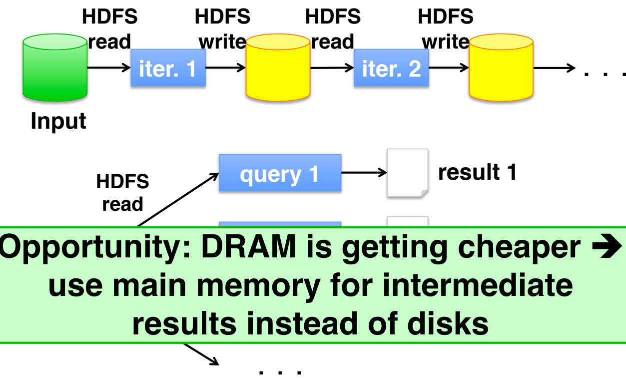
Spark Motivation

Complex jobs, interactive queries and online processing all need one thing that MR lacks:

Efficient primitives for **data sharing**

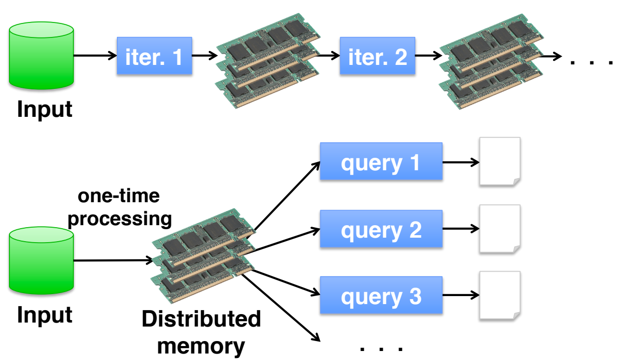
Problem: in MR, the only way to share data across jobs is using stable storage (e.g. file system) ➔ slow!

Examples



Opportunity: DRAM is getting cheaper ➔ use main memory for intermediate results instead of disks

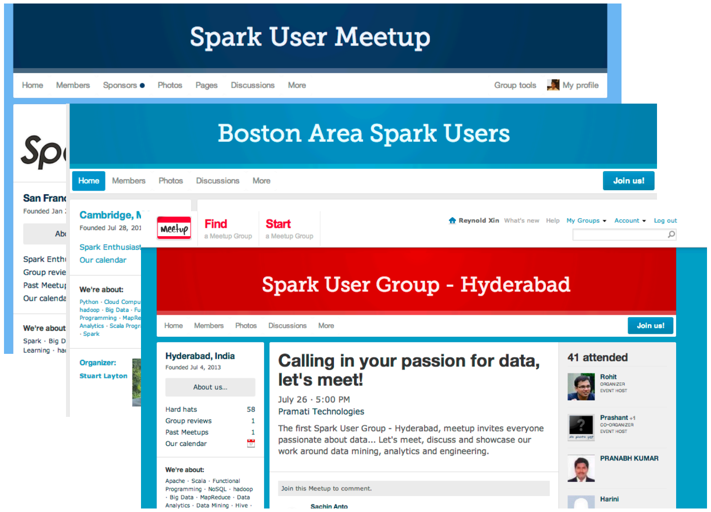
Goal: In-Memory Data Sharing



10-100× faster than network and disk

Solution: Resilient Distributed Datasets (RDDs)

- Partitioned collections of records that can be stored in memory across the cluster
- Manipulated through a diverse set of transformations (map, filter, join, etc)
- Fault recovery without costly replication
  - Remember the series of transformations that built an RDD (its lineage) to recompute lost data
- http://spark.incubator.apache.org/



Administrivia

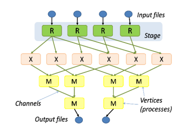
- Project 4
  - Design Doc due today (12/2) by 11:59pm
  - Code due next week Thu 12/12 by 11:59pm
- MIDTERM #2 is this Wednesday 12/4 5:30-7pm in 145 Dwinelle (A-L) and 2060 Valley LSB (M-Z)
  - Covers Lectures #13-24, projects, and readings
  - One sheet of notes, both sides
- Prof Joseph's office hours extended tomorrow:
  - 10-11:30 in 449 Soda
- RRR week office hours: TBA

5min Break

Datacenter Scheduling Problem

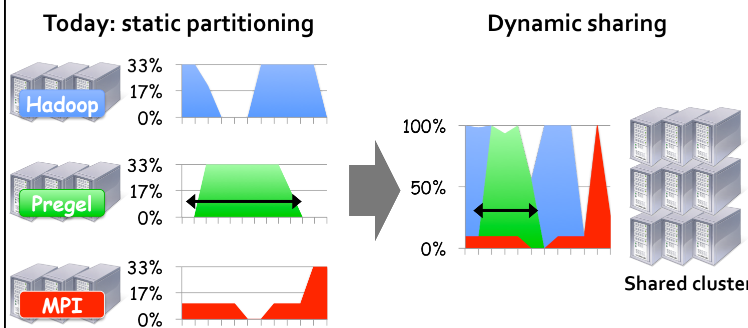
- Rapid innovation in datacenter computing frameworks
- **No single framework optimal for all applications**
- Want to run multiple frameworks in a single datacenter
  - …to maximize utilization
  - …to share data between frameworks

Hadoop MapReduce, Google Pregel, Pig, Rails



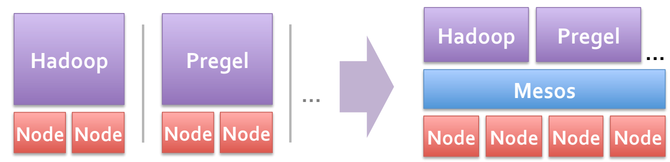
Dryad, CIEL, S4 distributed stream computing platform, Google Percolator, MPI2

Where We Want to Go



Solution: Apache Mesos

- Mesos is a common resource sharing layer over which diverse frameworks can run



- Run multiple instances of the *same* framework
  - Isolate production and experimental jobs
  - Run multiple versions of the framework concurrently
- Build *specialized frameworks* targeting particular problem domains
  - Better performance than general-purpose abstractions

Mesos Goals

- **High utilization** of resources
- **Support diverse frameworks** (current & future)
- **Scalability** to 10,000's of nodes
- **Reliability** in face of failures

http://mesos.apache.org/

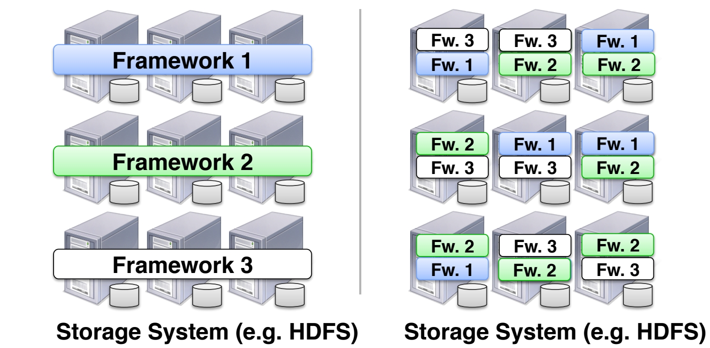
Resulting design: Small microkernel-like core that pushes scheduling logic to frameworks

Mesos Design Elements

- Fine-grained sharing:
  - Allocation at the level of *tasks* within a job
  - Improves utilization, latency, and data locality
- Resource offers:
  - Simple, scalable application-controlled scheduling mechanism

Element 1: Fine-Grained Sharing

Coarse-Grained Sharing (HPC): Fine-Grained Sharing (Mesos):



+ Improved utilization, responsiveness, data locality

Element 2: Resource Offers

- Option: Global scheduler
  - Frameworks express needs in a specification language, global scheduler matches them to resources

+ Can make optimal decisions

– Complex: language must support all framework needs

– Difficult to scale and to make robust

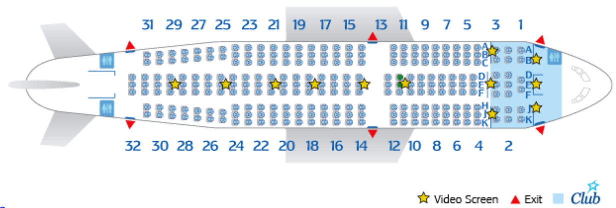
– Future frameworks may have unanticipated needs

Element 2: Resource Offers

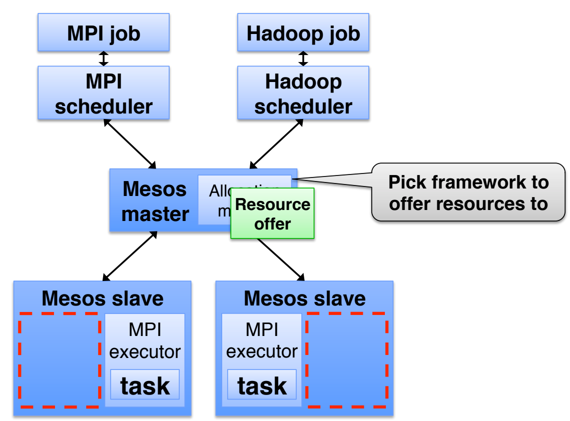
- Mesos: Resource offers
  - Offer available resources to frameworks, let them pick which resources to use and which tasks to launch

+ Keeps Mesos simple, lets it support future frameworks

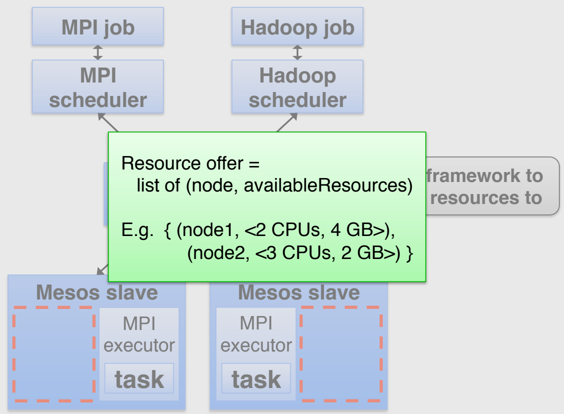
– Decentralized decisions might not be optimal



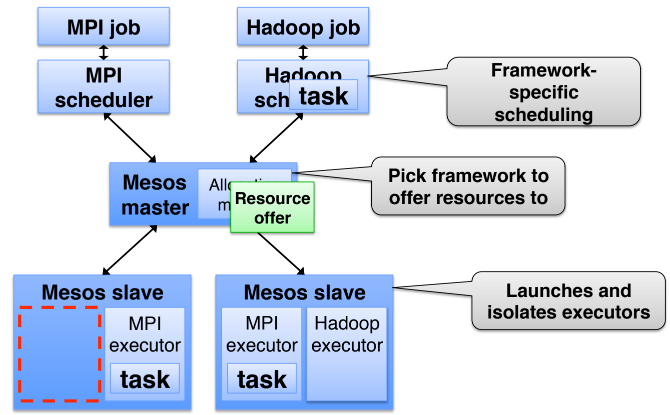
Mesos Architecture



Mesos Architecture



Mesos Architecture



Deployments

twitter: 1,000's of nodes running over a dozen production services

UCSF: Genomics researchers using Hadoop and Spark on Mesos

YAHOO!: Spark in use by Yahoo! Research

CONVIVA: Spark for analytics

Berkeley: Hadoop and Spark used by machine learning researchers

Summary

- Cloud computing/datacenters are the new computer
  - Emerging "Datacenter/Cloud Operating System" appearing
- Pieces of the DC/Cloud OS
  - High-throughput filesystems (GFS/HDFS)
  - Job frameworks (MapReduce, Apache Hadoop, Apache Spark, Pregel)
  - High-level query languages (Apache Pig, Apache Hive)
  - Cluster scheduling (Apache Mesos)