CS4740
CLOUD COMPUTING

Cloud Infrastructure in Industry

Prof. Chang Lou, UVA CS, Spring 2024
ARTIFICIAL INTELLIGENCE TOWN HALL
AN INTERDISCIPLINARY DISCUSSION
Ai and You: Navigating the Future Together at UVA and in the Broader World

EXPERT PANELISTS

PROF. MADHUR BEHL
Computer Science

PROF. REZA MOUSAVI
Commerce

PROF. MICHAEL PALMER
Center for Teaching Excellence, Director

TUESDAY
16th April, 2024
5:30 - 7:00 PM
NEWCOMB THEATRE
Located in Lower Level Below Pavilion XI

REGISTER NOW

Questions/Comments
sigai-contact@virginia.edu

sigai-at-uva.org
– We talked a lot about storage in this class, plus a bit about distributed computation. For storage, we focused on a particular type of interface (transactional databases).

– But there’s a vast range of infrastructural components that are needed for building successful distributed applications. Large companies and open-source communities have such components available.

– This lecture aims to provide an index of such components. We won’t give details about how these components are built, but pointers to where you can find out more.
  – the contents are heavily based on Malte Schwarzkopf’s talk at Cambridge
“WHAT IT TAKES TO BUILD GOOGLE?”
What happens here?
WHAT HAPPENS IN THOSE 139MS?
A TOUR TO DATACENTER

— 1. Datacenter hardware
— 2. Datacenter software
  — a. Google
  — b. Meta and Open source
  — c. Moving forward: ML stack
Hardware
From Meta (as of 2022):

- $O(?)$ machines in total
- $O(?)$ regions
- $O(?)$ interdependent services
- “Machine”
  - no chassis
  - DC battery
  - mostly custom-made
- Network
  - ToR switch
  - multi-path core
From Meta (as of 2022):
- \(O(1M)\) machines in total
- \(O(10s)\) regions
- \(O(1000s)\) interdependent services
- “Machine”
  - no chassis
  - DC battery
  - mostly custom-made
- Network
  - ToR switch
  - multi-path core
THE JOYS OF REAL HARDWARE

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures

slow disks, bad memory, misconfigured machines, flaky machines, etc.
WHAT IT TAKES TO MANAGE LARGE-SCALE SYSTEMS
and how is it different from HPC?

— Emphasis on commodity hardware
  — No expensive interconnect
  — Mid-range machines
  — Energy/performance/cost trade-off essential

— Massive automation
  — Very small number of on-site staff
  — Automated software bootstrap

— Fault tolerant design
  — Each component can fail
  — Software must be aware and compensate
Software
SOFTWARE SYSTEMS STACK

We’ll look at what goes here!

Linux kernel (customized)
Machine

Linux kernel (customized)
Machine

Linux kernel (customized)
Machine
THE Google STACK

data processing

**FlumeJava** [CRP+10] (parallel programming)
**Tenzing** [CLL+11] (SQL-on-MapReduce)
**MillWheel** [ABB+13] (stream processing)
**Pregel** [MAB+10] (graph processing)

**MapReduce** [DG08] (parallel batch processing)

**MegaStore** [BBC+11] (cross-DC ACID database)
**Spanner** [CDE+13] (cross-DC multi-version DB)
**Dremel** [MGL+10] (columnar database)

**GFS/Colossus** [GLG03] (distributed block store and file system)

**BigTable** [CDG+06] (row-consistent multi-dimensional sparse map)

**PowerDrill** [HBB+12] (query UI & columnar store)
**Dapper** [SBB+10] (pervasive tracing)

**Chubby** [Bur06] (locking and coordination)
**Borg** [VPK+15] and **Omega** [Ska+13] (cluster manager and job scheduler)

co-ordination & cluster management
### THE Google STACK

**Data Processing**

<table>
<thead>
<tr>
<th>Component</th>
<th>Method/Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FlumeJava</strong> [CRP+10]</td>
<td>parallel programming</td>
</tr>
<tr>
<td><strong>Tzening</strong> [CLL+11]</td>
<td>SQL-on-MapReduce</td>
</tr>
<tr>
<td><strong>MillWheel</strong> [ABB+13]</td>
<td>stream processing</td>
</tr>
<tr>
<td><strong>Pregel</strong> [MAB+10]</td>
<td>graph processing</td>
</tr>
</tbody>
</table>

**Data Storage**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MegaStore</strong> [BBC+11]</td>
<td>cross-DC ACID database</td>
</tr>
<tr>
<td><strong>Spanner</strong> [CDE+13]</td>
<td>cross-DC multi-version DB</td>
</tr>
<tr>
<td><strong>BigTable</strong> [CDG+06]</td>
<td>row-consistent multi-dimensional sparse map</td>
</tr>
<tr>
<td><strong>Dremel</strong> [MGL+10]</td>
<td>columnar database</td>
</tr>
</tbody>
</table>

**Monitoring**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dapper</strong> [SBB+10]</td>
<td>pervasive tracing</td>
</tr>
<tr>
<td><strong>CPI^2</strong> [ZTH+13]</td>
<td>interference mitigation</td>
</tr>
</tbody>
</table>

**Coordination & Cluster Management**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chubby</strong> [Bur06]</td>
<td>locking and coordination</td>
</tr>
<tr>
<td><strong>Borg</strong> [VPK+15] and <strong>Omega</strong> [SKA+13]</td>
<td>cluster manager and job scheduler</td>
</tr>
</tbody>
</table>
GFS/COLOSSUS

- Bulk data block storage system
  - Optimized for large files (GB-size)
  - Supports small files, but not common case
  - Read, write, record-append modes
- Colossus = GFSv2, adds some improvements
  - e.g., Reed-Solomon-based erasure coding
  - better support for latency-sensitive applications
  - sharded meta-data layer, rather than single master
MOTIVATION

— Lots of (semi-)structured data at Google
  — Web data: Contents, crawl metadata, links, anchors, pagerank, …
  — Per-user data: • User preference settings, recent queries/search results, …
  — Map data: • Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …

— Scale is huge
  — Billions of Web pages, many versions/page (~20K/version)
  — Hundreds of millions of users, thousands of q/sec
  — 100TB+ of satellite image data
  — (Above numbers are as of 2006-7!)
WEB SEARCH: THE COMPLETE WORKFLOW

Note: ignores page rank functionality for simplicity

GOALS

– Want asynchronous processes to continuously update different pieces of data
  – Want access to most current data at any time

– Need to support:
  – Very high read/write rates (millions of ops per second)
  – Efficient retrieval of small subsets of the data
  – Efficient scans over entire or subsets of the data

– Often want to examine data changes over time
  – E.g. Contents of a web page over multiple crawls
BIGTABLE (2006)

- ‘Three-dimensional‘ key-value store:
  - \(<\text{row key, column key, timestamp}> \rightarrow \text{value}\)
- Effectively a distributed, sorted, sparse map
SYSTEM ARCHITECTURE

Bigtable cell

Bigtable master
performs metadata ops, load balancing

Bigtable tablet server
serves data

Bigtable tablet server
serves data

Bigtable tablet server
serves data

GFS
holds tablet data

Chubby
holds metadata, handles master-election

Bigtable client
Bigtable client library

Metadata ops
Read/write
Open()
THE **Google** STACK


data processing

- **FlumeJava** [CRP+10] parallel programming
- **Tzening** [CLL+11] SQL-on-MapReduce
- **MillWheel** [ABB+13] stream processing
- **Pregel** [MAB+10] graph processing

- **MapReduce** [DG08] parallel batch processing
- **Percolator** [PD10] incremental processing
- **PowerDrill** [HBB+12] query UI & columnar store

- **MegaStore** [BBC+11] cross-DC ACID database
- **Spanner** [CDE+13] cross-DC multi-version DB
- **Dremel** [MGL+10] columnar database
- **Dapper** [SBB+10] pervasive tracing
- **CPI²** [ZTH+13] interference mitigation

- **BigTable** [CDG+06] row-consistent multi-dimensional sparse map

- **GFS/Colossus** [GLG03] distributed block store and file system

coordination & cluster management

- **Chubby** [Bur06] locking and coordination
- **Borg** [VPK+15] and **Omega** [SKA+13] cluster manager and job scheduler
SPANNER (2012)

- BigTable insufficient for some consistency needs
- Often have transactions across >1 data centers
  - May buy app on Play Store while travelling in the U.S.
  - Hit U.S. server, but customer billing data is in U.K.
  - Or may need to update several replicas for fault tolerance
- Wide-area consistency is hard
  - due to long delays and clock skew
  - no global, universal notion of time
  - NTP not accurate enough, PTP doesn’t work (jittery links)
SPANNER (2012)

- Spanner offers transactional consistency: full RDBMS
- Secret sauce: hardware-assisted clock sync
  - Using GPS and atomic clocks in data centres
- Use global timestamps and Paxos to reach consensus
  - Still have a period of uncertainty for write TX: wait it out!
  - Each timestamp is an interval:

```
<table>
<thead>
<tr>
<th>tt.earliest</th>
<th>t_{abs}</th>
<th>tt.latest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definitely in the past</td>
<td>t_{abs}</td>
<td>Definitely in the future</td>
</tr>
</tbody>
</table>
```
THE Google STACK


---

data processing

- **FlumeJava** [CRP+10] (parallel programming)
- **Tenzing** [CLL+11] (SQL-on-MapReduce)
- **MillWheel** [ABB+13] (stream processing)
- **Pregel** [MAB+10] (graph processing)

- **MapReduce** [DG08] (parallel batch processing)
- **Percolator** [PD10] (incremental processing)
- **PowerDrill** [HBB+12] (query UI & columnar store)

- **MegaStore** [BBC+11] (cross-DC ACID database)
- **Spanner** [CDE+13] (cross-DC multi-version DB)
- **Dremel** [MGL+10] (columnar database)
- **Dapper** [SBB+10] (pervasive tracing)
- **CPI²** [ZTH+13] (interference mitigation)

- **GFS/Colossus** [GL03] (distributed block store and file system)

---

coordination & cluster management

- **Chubby** [Bur06] (locking and coordination)
- **Borg** [VPK+15] and **Omega** [SKA+13] (cluster manager and job scheduler)
MAPREDUCE (2004)

- Parallel programming framework for scale
  - Run a program on 100’s to 10,000’s machines
- Framework takes care of:
  - Parallelization, distribution, load-balancing, scaling up (or down) & fault-tolerance
- Accessible: programmer provides two methods ;-) 
  - map(key, value) → list of <key’, value’> pairs
  - reduce(key’, value’) → result
- Inspired by functional programming
MAPREDUCE (2004)

Input

Map

Perform Map() query against local data matching input specification

X: 5

X: 3

Y: 1

Y: 7

Shuffle

Aggregate gathered results for each intermediate key using Reduce()

Reduce

X: 8

Y: 8

Output

End user can query results via distributed key/value store

Results: X: 8, Y: 8
MAPREDUCE: PROS & CONS

– Extremely simple, and:
  – Can auto-parallelize (since operations on every element in input are independent)
  – Can auto-distribute (since rely on underlying Colossus/BigTable distributed storage)
  – Gets fault-tolerance (since tasks are idempotent, i.e. can just re-execute if a machine crashes)

– Doesn’t really use any sophisticated distributed systems algorithms (except storage replication)

– However, not a panacea:
  – Limited to batch jobs, and computations which are expressible as a map() followed by a reduce()
DREMEL (2010)

— Column-oriented store
   — For quick, interactive queries

Row-oriented storage

Column-oriented storage
DREMELE (2010)

<table>
<thead>
<tr>
<th>user_id</th>
<th>user_name</th>
<th>current_balance</th>
<th>number_of_transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'<a href="mailto:freddie@gmail.com">freddie@gmail.com</a>'</td>
<td>1059298</td>
<td>1224</td>
</tr>
<tr>
<td>2</td>
<td>'<a href="mailto:lindsey@gmail.com">lindsey@gmail.com</a>'</td>
<td>254</td>
<td>1045</td>
</tr>
<tr>
<td>3</td>
<td>'<a href="mailto:tabby@yahoo.com">tabby@yahoo.com</a>'</td>
<td>3910</td>
<td>194</td>
</tr>
<tr>
<td>4</td>
<td>'<a href="mailto:philip@hotmail.com">philip@hotmail.com</a>'</td>
<td>234028</td>
<td>130</td>
</tr>
<tr>
<td>5</td>
<td>'<a href="mailto:elon@x.com">elon@x.com</a>'</td>
<td>-440000000000</td>
<td>1</td>
</tr>
</tbody>
</table>

SELECT sum(current_balance) 
FROM table
WHERE user_id > 2

SELECT user_id, user_name, 
current_balance
FROM table
WHERE user_id = 1

Suitable for columnar DB
Suitable for row-oriented DB
CHUBBY SUMMARY

- Lock Service
- Chubby uses Paxos for everything
  - Propagate writes to a file
  - Choosing a Master
  - Even for adding new Chubby servers to a Chubby cell
- Paxos transforms a multi-node service into something that looks very much like one fault-tolerant, albeit slower, server! -> pretty close to distributed systems’ core goal
CHUBBY INTERFACE: UNIX FILE SYSTEM

– Chubby supports a strict tree of files and directories
  – The way to think about these files is that they are locks with a little bit of contents (e.g., identity and location of a primary)
  – No symbolic links, no hard links
  – /ls/foo/wombat/pouch
    – 1st component (ls): lock service (common to all names)
    – 2nd component (foo): the chubby cell (used in DNS lookup to find the cell master)
    – The rest: name inside the cell

– Support most normal operations
  – Create, delete, open, write, ...

– Support reader/writer lock on a node
EXAMPLE: PRIMARY ELECTION

Open("/ls/foo/OurServicePrimary", "write mode");
if (successful) {
    // primary
    SetContents(primary_identity);
} else {
    // replica
    Open("/ls/foo/OurServicePrimary", "read mode",
         "file-modification event");
    when notified of file modification:
    primary = GetContentsAndStat();
}
BORG

— Cluster manager and scheduler
  — Tracks machine and task liveness
  — Decides where to run what
— Consolidates workloads onto machines
  — Efficiency gain, cost savings
  — Need fewer clusters
— You might be more familiar with its successor:
BACKGROUND: CONTAINERS

KUBERNETES ARCHITECTURE

Kubernetes cluster

Control Plane

Node

Cloud provider API

API server

Cloud controller manager (optional)

Controller manager

etcd (persistence store)

kubelet

kube-proxy

Scheduler

Control plane

Node
THE Meta STACK


parallel data processing

**Hive** [TSA⁺10]  
SQL-on-MapReduce

**Peregrine** [MG12]  
interactive querying

**Scuba** [AAB⁺13]  
in-memory database

**ÜberTrace** [CMF⁺14, §3]  
pervasive tracing

(Hadoop) **MapReduce** [DG08]  
parallel batch processing

**Unicorn** [CBB⁺13]  
graph processing

**MysteryMachine** [CMF⁺14]  
performance modeling

monitoring tools

**Haystack** [BKL⁺10]  
hot blob storage

**TAO** [BAC⁺13]  
graph store

**Wormhole** [SAA⁺15]  
pub-sub replication

**f4** [MLR⁺14]  
warm blob storage

**memcached** [NFG⁺13]  
in-memory key-value store/cache

**HBase** [BGS⁺11]  
multi-dimensional sparse map

**MySQL**  
sharded ACID database

**Bistro** [GSW15]  
cluster scheduler

**HDFS** [SKR⁺10]  
distributed block store and file system

**data storage**
THE Meta STACK

parallel data processing

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL-on-MapReduce</td>
<td>interactive querying</td>
<td>in-memory database</td>
<td>pervasive tracing</td>
</tr>
<tr>
<td>(Hadoop) MapReduce [DG08]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

parallel batch processing

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hot blob storage</td>
<td>graph store</td>
<td>pub-sub replication</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>multi-dimensional sparse map</td>
<td>warm blob storage</td>
<td>in-memory key-value store/cache</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>distributed block store and file system</td>
<td>sharded ACID database</td>
<td>cluster scheduler</td>
</tr>
</tbody>
</table>

data storage

monitoring tools

<table>
<thead>
<tr>
<th>Mystery Machine [CMF+14]</th>
</tr>
</thead>
<tbody>
<tr>
<td>performance modeling</td>
</tr>
</tbody>
</table>

HAYSTACK & F4

- Blob stores, hold photos, videos
  - not: status updates, messages, like counts
- Items have a level of hotness
  - How many users are currently accessing this?
  - Baseline “cold” storage: MySQL
- Want to cache close to users
  - Reduces network traffic
  - Reduces latency
  - But cache capacity is limited!
  - Replicate for performance, not resilience
What about other companies’ stacks?
HOW ABOUT OTHER COMPANIES?

— Very similar stacks.
  — Microsoft, Yahoo, Twitter all similar in principle.
— Typical set-up:
  — Front-end serving systems and fast back-ends.
  — Batch data processing systems.
  — Multi-tier structured/unstructured storage hierarchy.
  — Coordination system and cluster scheduler.
— Minor differences owed to business focus
  — e.g., Amazon focused on inventory/shopping cart.
OPEN SOURCE SOFTWARE

— Lots of open-source implementations!
  — MapReduce → Hadoop, Spark, Metis
  — GFS → HDFS
  — BigTable → HBase, Cassandra
  — Borg → Mesos, Firmament
  — Chubby → Zookeeper

— But also some releases from companies...
  — Presto (Facebook)
  — Kubernetes (Google Borg)
THE Spark STACK

- **Spark Streaming**
  - Stream processing

- **GraphX**
  - Graph computation

- **MLlib**
  - User-friendly machine learning

- **SparkSQL**
  - SQL API

- **BlinkDB**
  - SQL w/ bounded errors/response times

- **Spark**
  - Fast memory-optimized execution engine (Python/Java/Scala APIs)

- **Tachyon**
  - Distributed Memory-Centric Storage System

- **Hadoop Distributed File System (HDFS)**

- **Mesos**
  - Cluster resource manager, multi-tenancy

- **Hadoop MR**

- **Hive**

- **Storm**

- **MPI**

*Supported Release*, *In Development*, *Related External Project*
NEWER STACKS

— Lots of new support for machine learning
  — Google: Tensorflow, Tensorflow Serving, Tensorflow Extended (TFX)
  — Uber: Michelangelo
  — Spark/Berkeley Data Stack (BDAS): MLBase, MLlib, Clipper
HEWLETT-PACKARD (HP)
MLOPS INFRASTRUCTURE & TOOLING
https://fullstackdeeplearning.com/spring2021/lecture-6/
TAKEAWAYS

– Running at huge (10k+ machines) scale requires different software stacks.
– Pretty interesting systems and design challenges.
  – try to read more papers! (e.g., BigTable, Spanner..)
– Emerging new support for ML workloads.
– Next class: Lab Day II, Hack ZooKeeper
REFERENCES


ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.