CS349D Cloud Computing

Christos Kozyrakis & Matei Zaharia

Fall 2017, 10:30–12:00, 380-380W

http://cs349d.stanford.edu

Class Staff

Christos Kozyrakis

http://www.stanford.edu/~kozyraki

Matei Zaharia

https://cs.stanford.edu/~matei

James Thomas (TA)

http://cs.stanford.edu/~jjthomas

Topics

Cloud computing overview

Cloud economics (2)

Storage

Databases

Serverless computing

Analytics & streaming systems

Security & privacy

Debugging & monitoring

Resource allocation

Operations

Serving systems

Programming models

ML as a service

Hardware acceleration

CAP theorem

Class Format

- One topic per class meeting
  - We all read the paper ahead of time
  - Submit answer to 1-2 questions before meeting
  - 1-2 students summarize paper & lead discussion
  - <u>We all participate actively in the discussion</u>
  - 1 student keeps notes
- A few guest lectures
  - See schedule online

What to Look for in a Paper

- The challenge addressed by the paper
- The key insights & original contributions
  - Real or claimed, you have to check
- Critique: the major strengths & weaknesses
  - Look at the claims and assumptions, the methodology, the analysis of data, and the presentation style
- Future work: extensions or improvements
  - Can we use a similar methodology to other problems?
  - What are the broader implications?

Tips for Reading Papers

Read the abstract, intro, & conclusions sections first

Read the rest of the paper twice

First a quick pass to get rough idea then a detailed reading

Underline/highlight the important parts of the paper

Keep notes on the margins about issues/questions

Important insights, questionable claims, relevance to other topics, ways to improve some technique etc.

Look up references that seem to important or missing

You may also want to check who and how references this paper

Research Project

Groups of 2-3 students

Topic

- Address an open question in cloud computing
- Suggested by staff or suggest your own

Timeline

- Project proposal – October 9th
- Mid-quarter checkpoint – November 6th
- Presentation/paper – week of December 3rd

Reminders

- Make sure you are registered on Axess
  - Contact instructors for access code
- Sign up to lead a discussion topic
  - We will assign topics for note taking
- Start talking about projects
  - Form a team

Cloud Computing Overview

Christos Kozyrakis & Matei Zaharia

http://cs349d.stanford.edu

What is Cloud Computing?

Informal: computing with large datacenters

What is Cloud Computing?

~~Informal: computing with large datacenters~~

Our focus: computing as a utility

- » Outsourced to a third party or internal org

Types of Cloud Services

Infrastructure as a Service (IaaS): VMs, disks

Platform as a Service (PaaS): Web, MapReduce

Software as a Service (SaaS): Email, GitHub

Public vs private clouds:

Shared across arbitrary orgs/customers
vs internal to one organization

Example

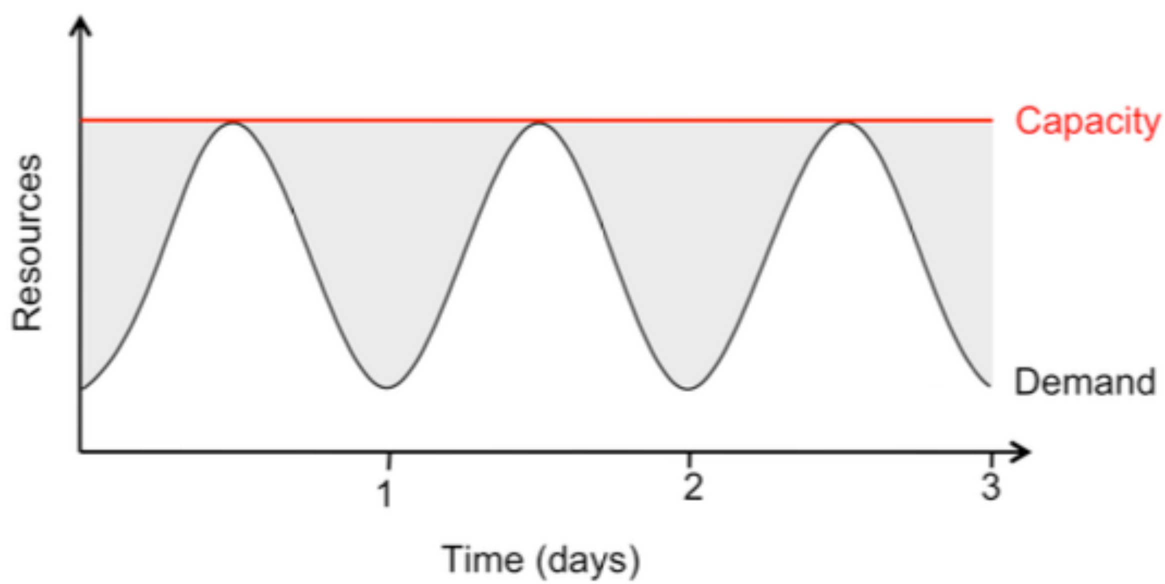
AWS Lambda functions-as-a-service

- Runs functions in a Linux container on events
- Used for web apps, stream processing, highly parallel MapReduce and video encoding

Cloud Economics: For Users

Pay-as-you-go (usage-based) pricing:

- » Most services charge per minute, per byte, etc
- » No minimum or up-front fee
- » Helpful when apps have *variable utilization*

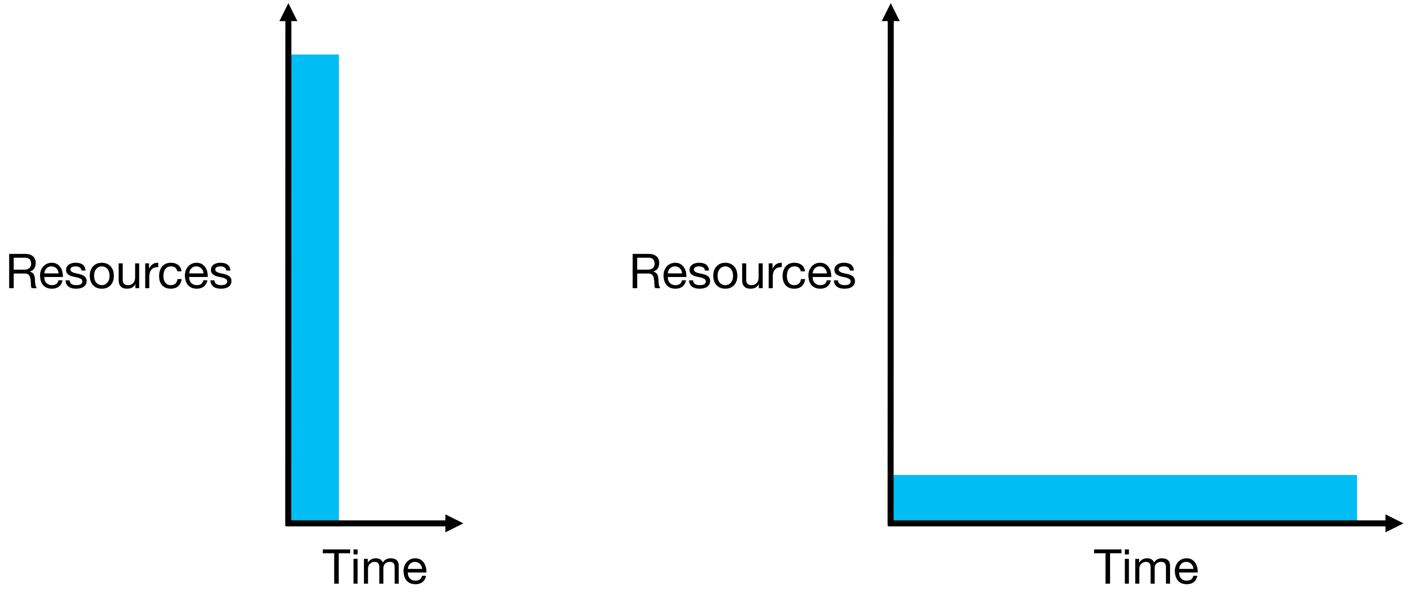


Cloud Economics: For Users

Elasticity:

» Using 1000 servers for 1 hour costs the same as 1 server for 1000 hours

» Same price to get a result faster!



Cloud Economics: For Providers

Economies of scale:

» Purchasing, powering, managing machines at scale gives lower per-unit costs than customers'

Other Interesting Features

Spot market for preemptible machines

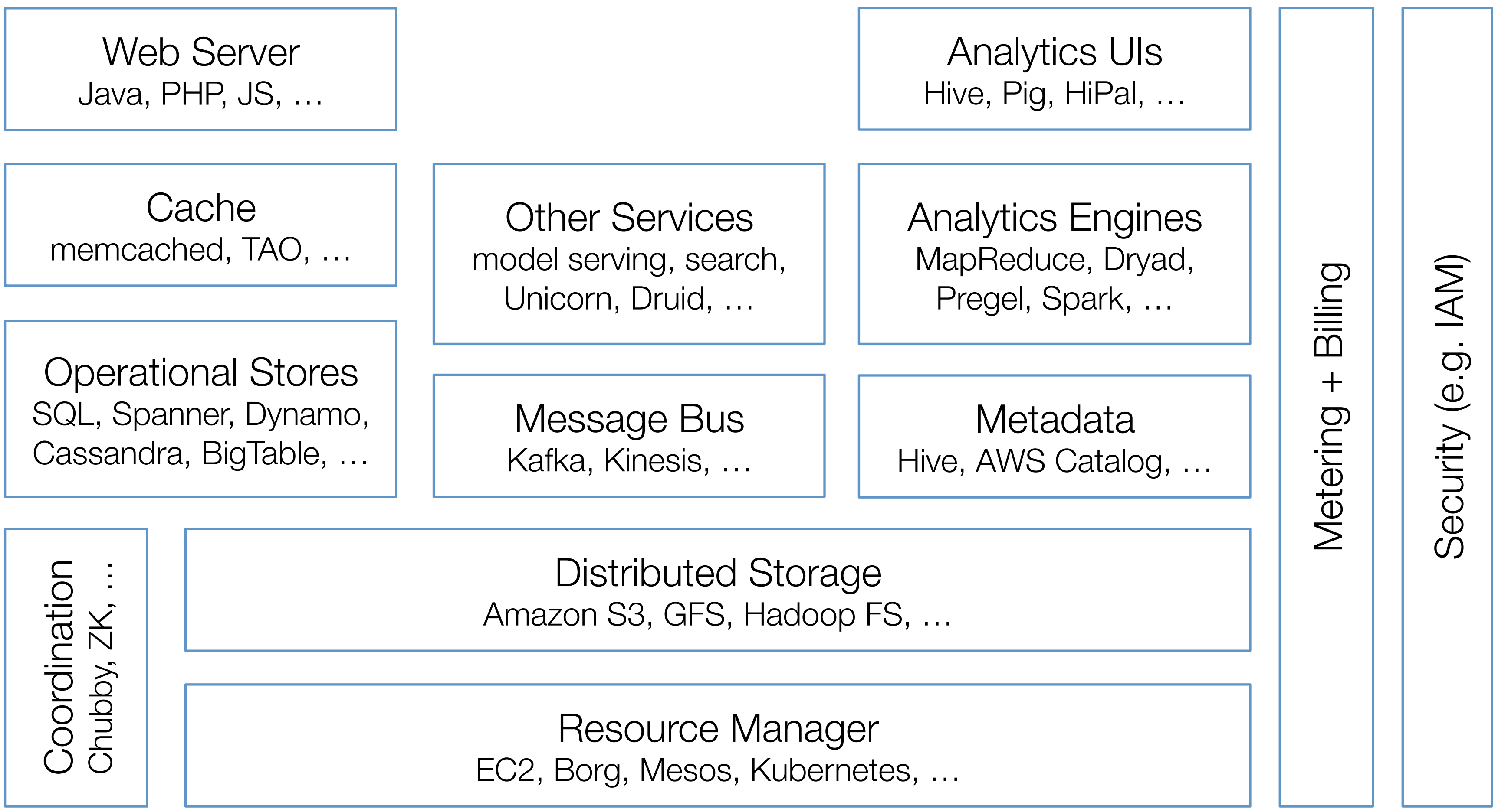
Reserved instances and RI market

Ability to quickly try exotic hardware

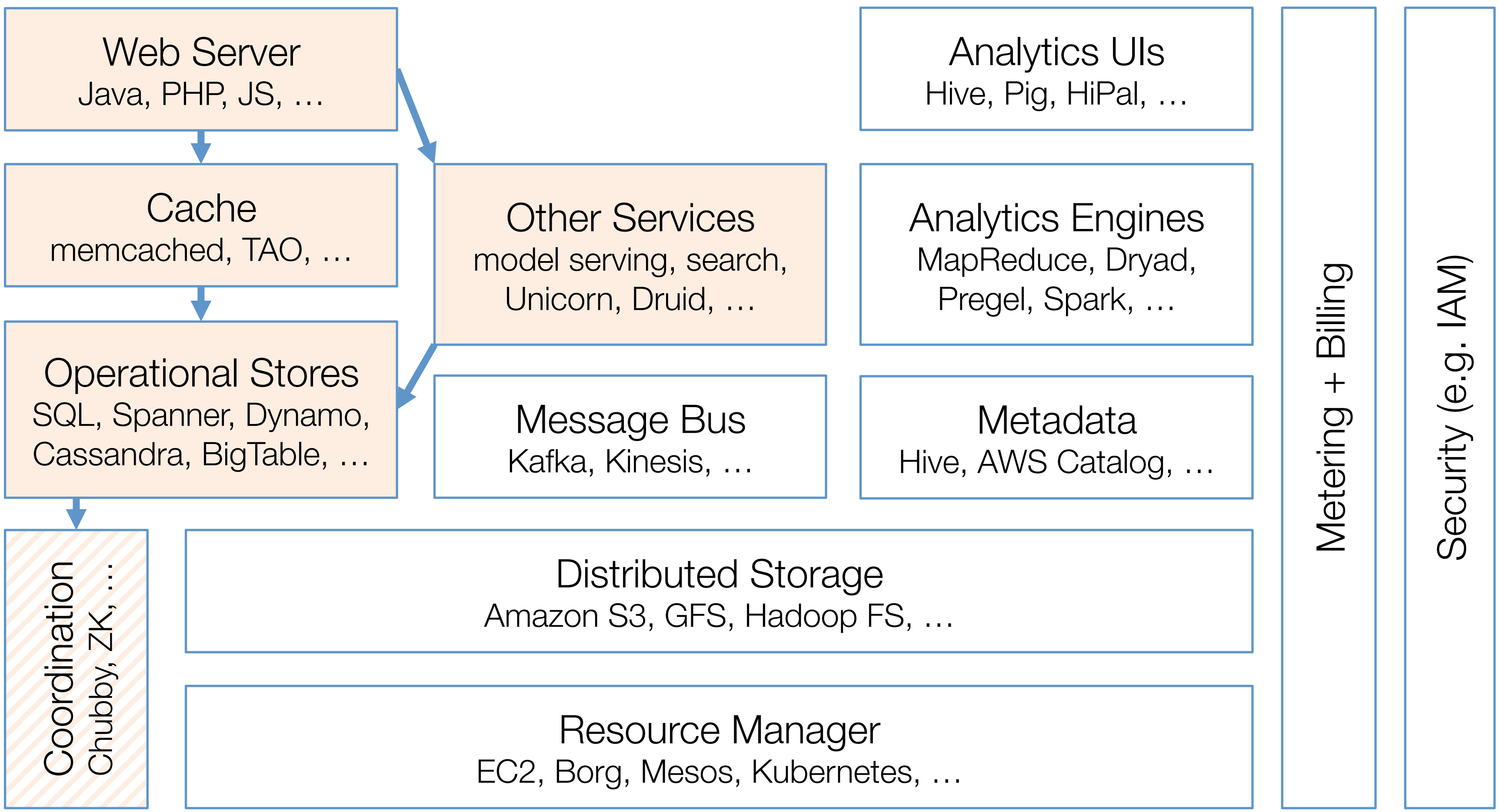
Common Cloud Applications

1. Web/mobile applications
2. Data analytics (MapReduce, SQL, ML, etc)
3. Stream processing
4. Batch computation (HPC, video, etc)

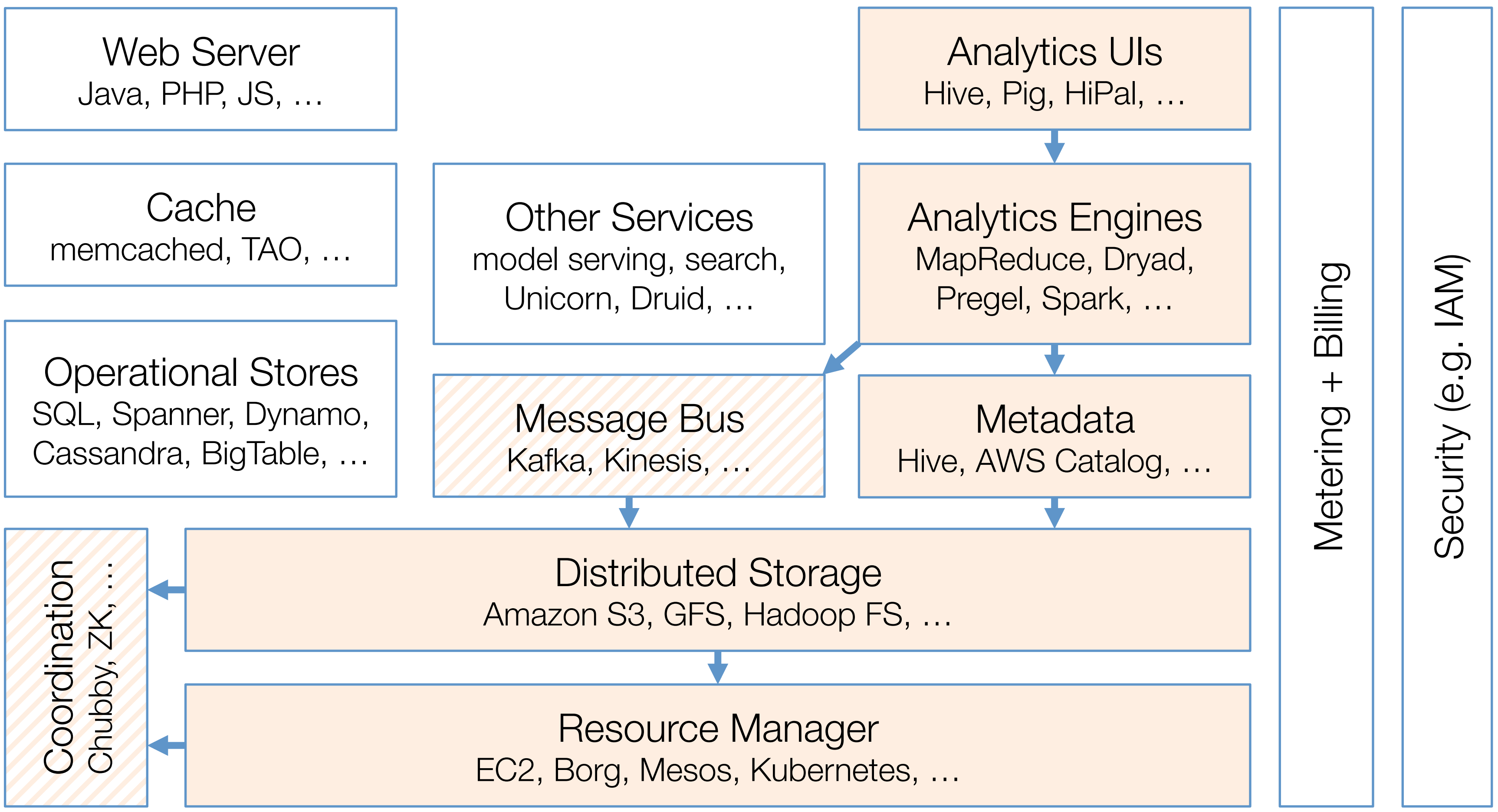
Cloud Software Stack



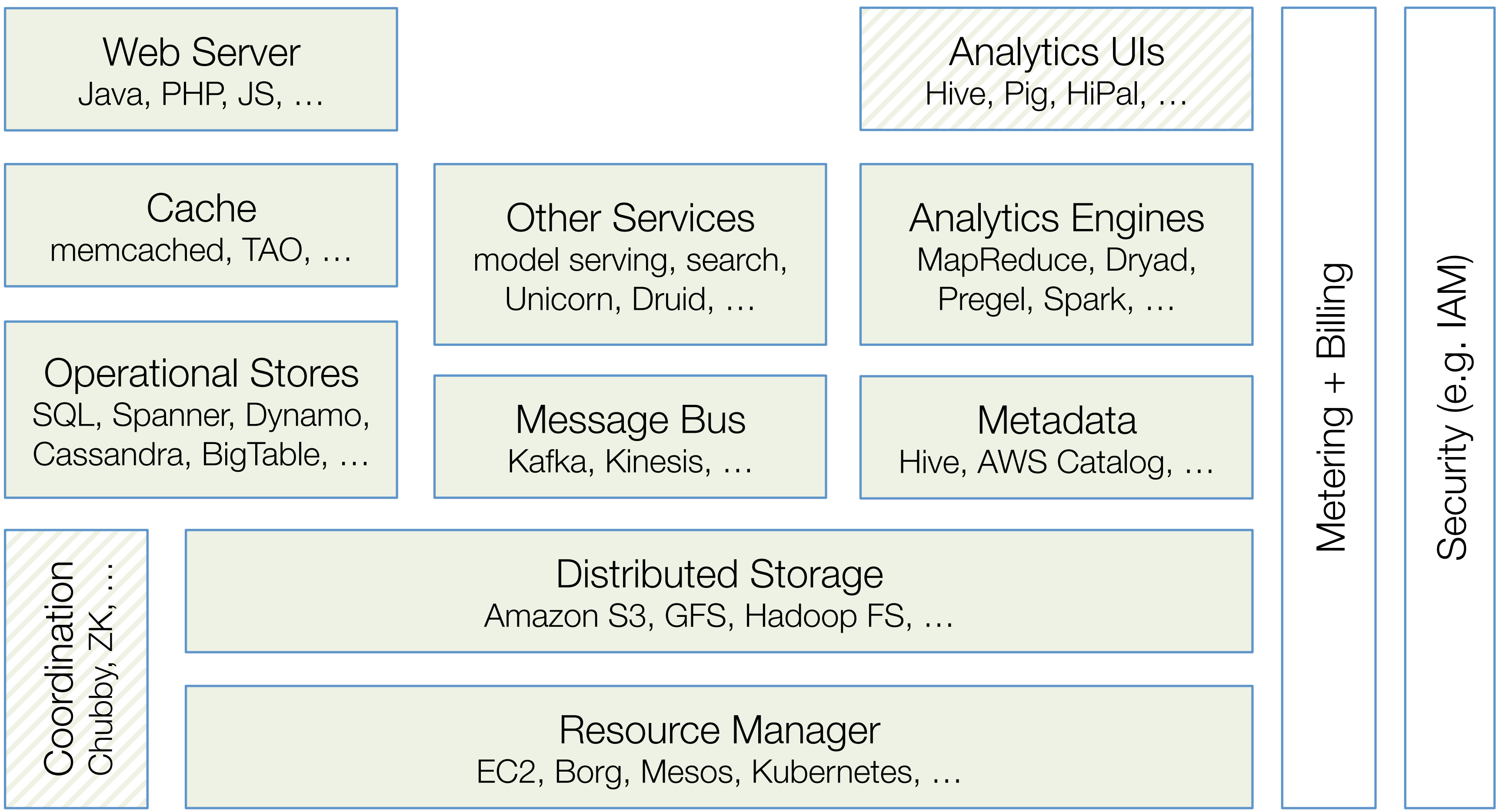
Example: Web Application



Example: Analytics Warehouse

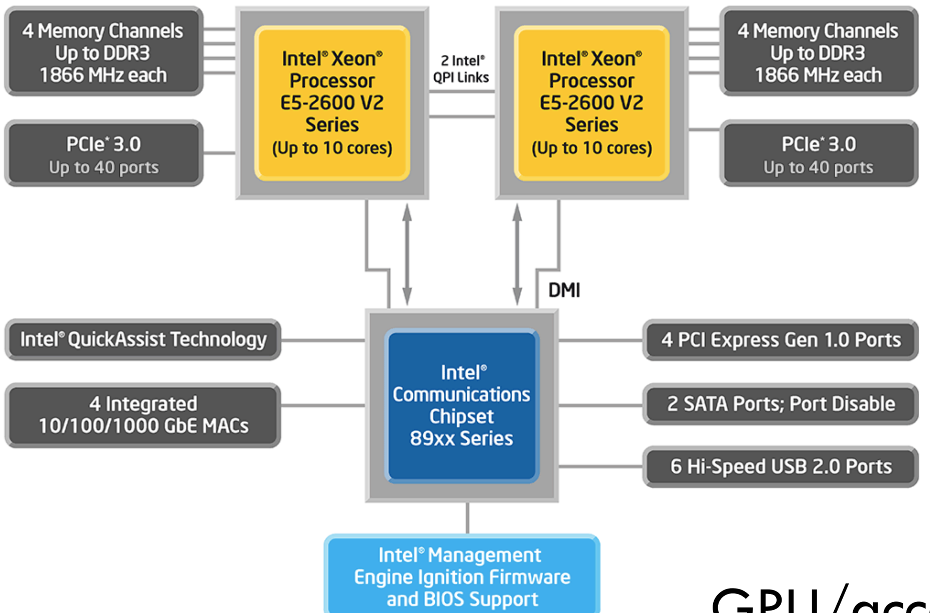


Components Offered as PaaS

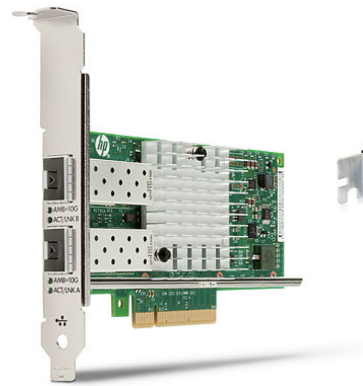


Datacenter Hardware

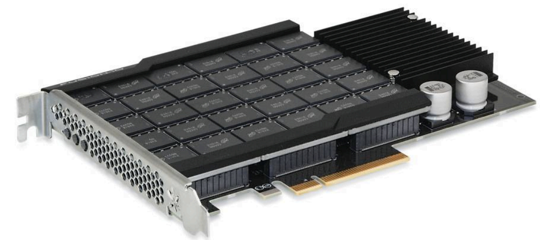
2-socket server



>10GbE NIC



Flash Storage

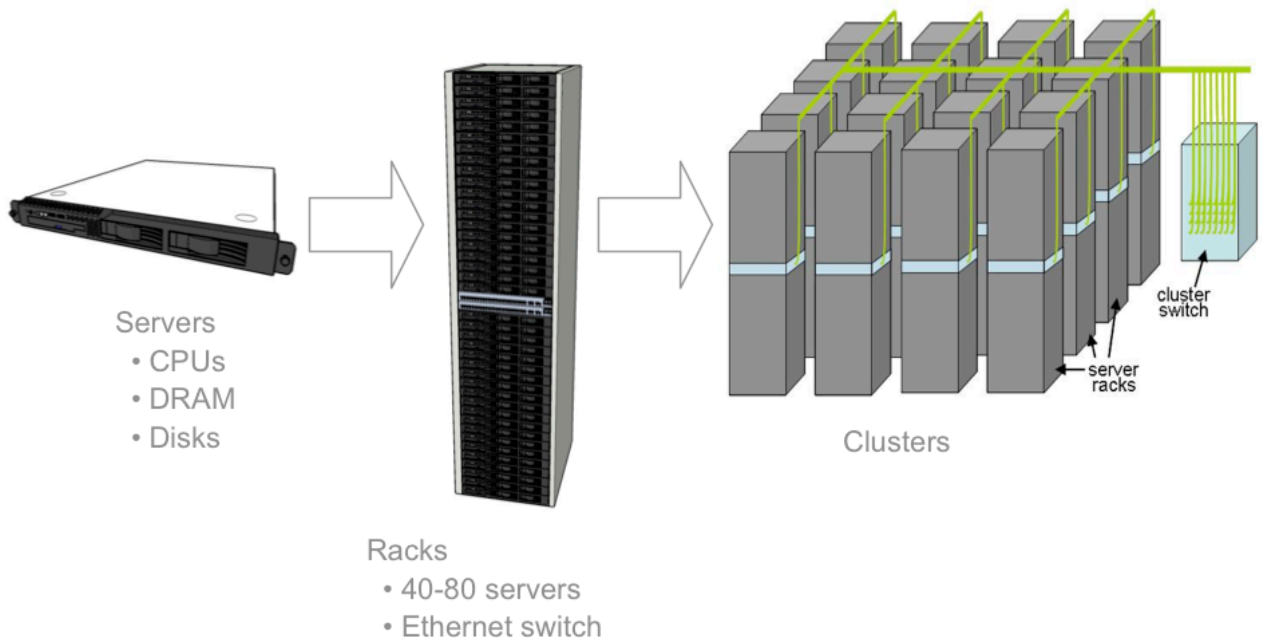


JBOD disk array

GPU/accelerators

>10GbE Switch

Datacenter Hardware



Rows of rack-mounted servers

Datacenters with 50 – 200K of servers and burn 10 – 100MW

Storage: distributed with compute or NAS systems

Remote storage access for many use cases (why?)

Hardware Heterogeneity

Standard Systems	I Web	III Database	IV Hadoop	V Haystack	VI Feed
CPU	High 2 x E5-2670	High 2 x E5-2660	High 2 x E5-2660	Low 1 x E5-2660	High 2 x E5-2660
Memory	Low 16GB	High 144GB	Medium 64GB	Med-Hi 96GB	High 144GB
Disk	Low 250GB	High IOPS 3.2 TB Flash	High 15 x 4TB SATA	High 30 x 4TB SATA	Medium 2TB SATA + 1.6TB Flash
Services	Web, Chat	Database	Hadoop	Photos, Video	Multifeed, Search, Ads

[Facebook server configurations]

Custom-design servers

- Configurations optimized for major app classes
- Few configurations to allow reuse across many apps
- Roughly constant power budget per volume

Useful Latency Numbers

Initial list from Jeff Dean, Google

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L3 cache reference	20 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Snappy	3,000 ns
Send 2K bytes over 10Ge	2,000 ns
Read 1 MB sequentially from memory	100,000 ns
Read 4KB from NVMe Flash	50,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA → Europe → CA	150,000,000 ns

Useful Throughput Numbers

DDR4 channel bandwidth	20 GB/sec
PCIe gen3 x16 channel	15 GB/sec
NVMe Flash bandwidth	2GB/sec
GbE link bandwidth	10 – 100 Gbps
Disk bandwidth	6 Gbps
NVMe Flash 4KB IOPS	500K – 1M
Disk 4K IOPS	100 – 200

Performance Metrics

Throughput

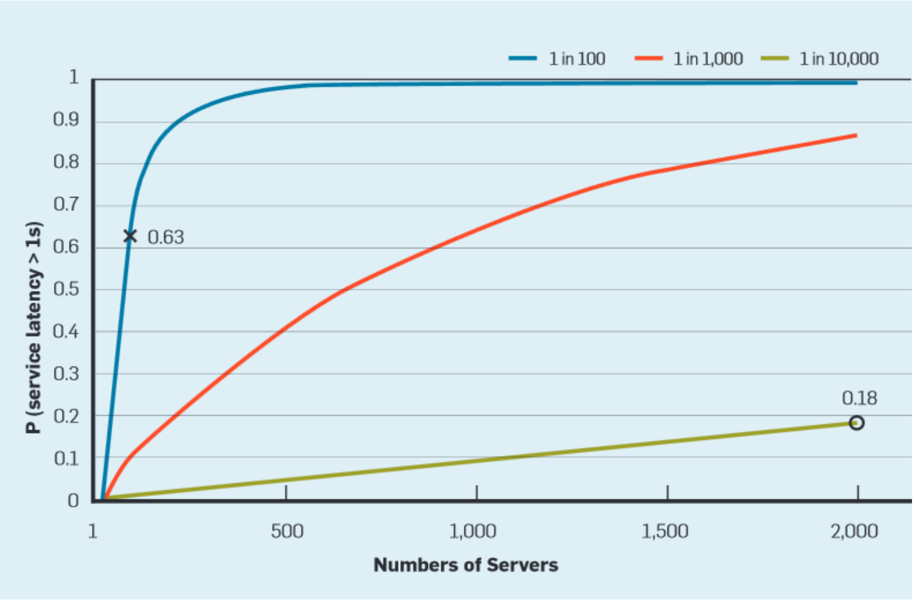
- Requests per second
- Concurrent users
- Gbytes/sec processed
- ...

Latency

- Execution time
- Per request latency

Tail Latency

[Dean & Barroso,'13]



The 95th or 99th percentile request latency

End-to-end with all tiers included

Larger scale → more prone to high tail latency

Total Cost of Ownership (TCO)

TCO = capital (CapEx) + operational (OpEx) expenses

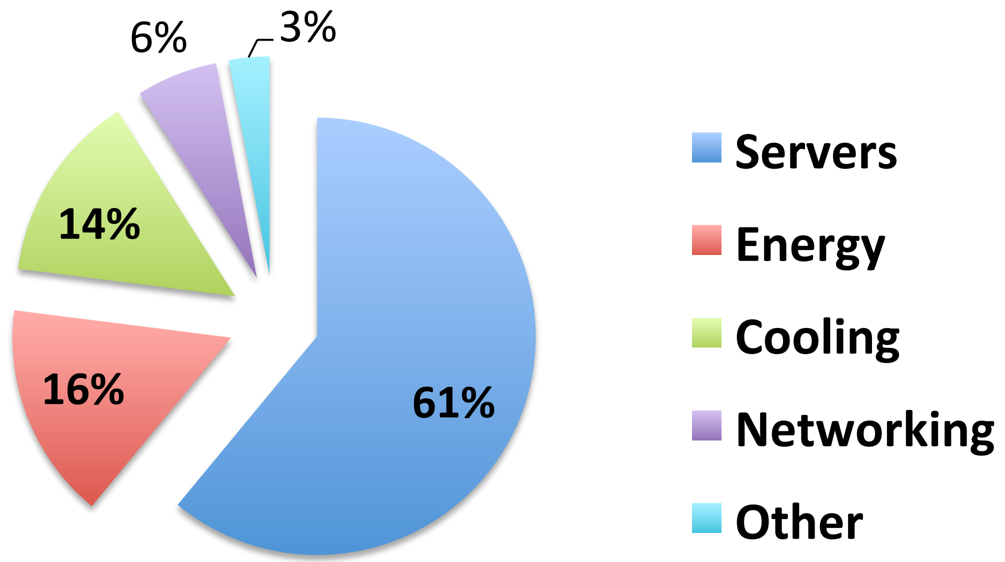
Operators perspective

- CapEx: building, generators, A/C, compute/storage/net HW
  - Including spares, amortized over 3 – 15 years
- OpEx: electricity (5-7c/KWh), repairs, people, WAN, insurance, …

Users perspective

- CapEx: cost of long term leases on HW and services
- OpeEx: pay per use cost on HW and services, people

Operator's TCO Example



[Source: James Hamilton]

Hardware dominates TCO, make it cheap

Must utilize it as well as possible

Reliability

Failure in time (FIT)

- Failures per billion hours of operation = 10^9/MTTF

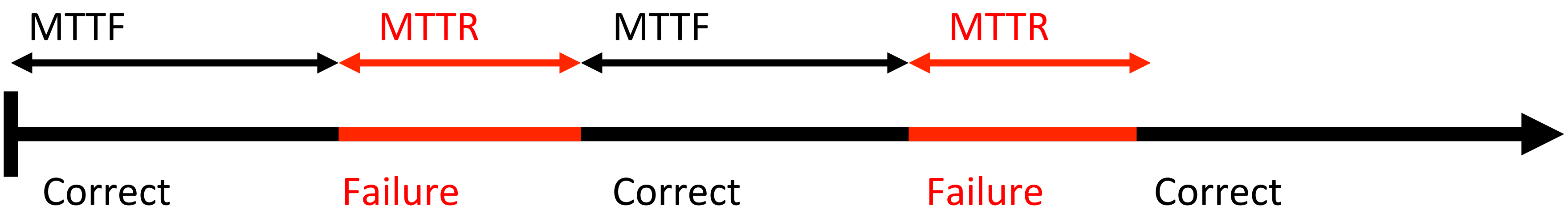
Mean time to failure (MTTF)

- Time to produce first incorrect output

Mean time to repair (MTTR)

- Time to detect and repair a failure

Availability



Steady state availability = MTTF / (MTTF + MTTR)

Yearly Datacenter Flakiness

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)

~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hrs to come back)

~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hrs)

~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% packet loss)

~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 (2-4% failure rate, machines crash at least twice)

~thousands of hard drive failures (1-5% of all disks will die)

Add to these SW bugs, config errors, human errors, …

Key Availability Techniques

Technique	Performance	Availability
Replication	✔	✔
Partitioning (sharding)	✔	✔
Load-balancing	✔	
Watchdog timers		✔
Integrity checks		✔
Canaries		✔
Eventual consistency	✔	✔

Make apps do something reasonable when not all is right

- Better to give users limited functionality than an error page

Aggressive load balancing or request dropping

- Better to satisfy 80% of the users rather than none

The CAP Theorem

In distributed systems, choose 2 out of 3

Consistency

Every read returns data from most recent write

Availability

Every request executes & receives a (non-error) response

Partition-tolerance

The system continues to function when network partitions occur (messages dropped or delayed)

Useful Tips

Check for single points of failure

Keep it simple stupid (KISS)

- The reason many systems use centralized control

If it’s not tested, do no rely on it

Question: how do you test availability techniques with hundreds of loosely coupled services running on thousands of machines?