Big Data Technology
System Design Principles

Eshcar Hillel
Yahoo!

Ronny Lempel
Outbrain
Data = Systems

We need to **Move**, **Store** and **Process** data

Big Data = Big Systems
How to Get the Big Systems Right?

- A multidisciplinary science on its own right
  - Distributed Computing, Networking
  - Hardware and Software Architecture
  - Operations Research, Measurement, Performance Evaluation
  - Power Management
  - ... and even Civil Engineering

- In this course - aspects related to Computer Science

- We’ll start with some principles ...
  - And see how they manifest in real systems
Principles of “Big Data” Systems

- Scalability - handle a growing amount of data
  - Elasticity – adding more computational resources deals seamlessly & transparently with the growth of data/users
- Resiliency \( \sim \) redundancy
  - Fault-tolerance
  - Tail (-latency) tolerant
- Goal-oriented design/workload optimization:
  - What matters more, latency or throughput?
  - What are the access patterns to the data?
  - Is the task I/O bound or CPU bound?
  - Consistency: stronger (strict) requirements (financial transactions)? or perhaps best-effort is enough?
An Ideal System Should ...

1. Scale
Architect’s Dream - Throughput

How many requests can be served in a unit of time?
Architect’s Dream - Latency

How long does a single request take?
Throughput-Latency Graphs

- Latency percentiles
- "knee" = Saturation point
- 5-50 threads

Write only workload

Latency ms - log scale

Throughput MB/s

CS236620 Big Data Technology
Scaling Vertically (Up)?
Scaling Horizontally (Out)?
Example: Network Filesystems

Monolithic
(e.g., historical NFS)

Distributed
(e.g., Hadoop FS)
Scale-Up Philosophy

- A basic building block for a distributed system
  - Utilizes multicores CPUs
  - Exploits highly concurrent in-memory data structures
- Minimize centralized processing
  - Shared counter for snapshot consistency semantics
- Maximize parallelism
  - Operations can proceed concurrently
Scale-Out Philosophy

- Partitioning with centralized management
- Scalability through Decoupling
  - HDFS: Metadata and Data accesses decoupled
  - Whatever is split can be scaled independently
- Minimize centralized processing
  - Metadata accesses coordinated but lean
- Maximize I/O parallelism
  - Clients access the data nodes concurrently
The Peer-to-Peer Approach

- Completely server-less
  - All nodes and functions are **fully symmetric**
  - E.g., in a distributed data store every node has a serving function and a management function

- Less favored in managed DC environments
  - Very hard to maintain consistency guarantees
  - Very hard to optimize globally
  - Lightweight centralized critical services prevail
Elasticity

- Resource demands often unknown in advance
  - Driven by application popularity
- Goal: enablement of organic growth
  - Add- (and pay-) as-you-grow

- **Economies of scale**
  - Pool multiple datasets and services in huge DC’s
  - Better use of shared resources (personnel, real estate, electricity, network, compute and storage)
An Ideal System Should ...

2. Be Resilient
The Joys of Real Hardware (Jeff Dean - Ladis 2009)

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.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.
Resilience = Redundancy

- Redundant copies of all data files on disk
- Storage of intermediate processing results on disk
- Automatic detection of node or processing failures and re-mirroring
- Selective re-computation of results
Protecting the Critical Services
Replicated Metadata Server

Operates as Primary-Backup process pair

- Backup becomes primary upon failure
- Recovers state from a shared consistent log
Classic Problem: Split Brain

Failure detection can’t be perfect [theory!]

The primary and the backup might end up working concurrently

... competing for access to the (persistent) metadata log

ZooKeeper helps avoiding this problem
Electing a Master

- Both suggest to become masters
- Elect smallest id
- Process 0 elected as master
Electing a Master

Processes are asynchronous

- Messages can be slow
- Or processes are partitioned

Both processes are elected

- Undesirable!

Process 1

Process 0

Process 0 can lead

Process 1 can lead
Electing a Master w/ ZooKeeper

- Zookeeper deals with concurrent requests
- Consistent independent of partitions and speed of messages
Apache ZooKeeper

- A configuration and coordination service for distributed systems
  - Leader election
  - Group membership
  - Metadata/configuration management
- Exposes API for managing a hierarchical name space (znode files)
  - Manipulate znodes: create, exists, delete, getChildren, ...
  - Watches, notifications, versions
- Encapsulates consensus protocol (agreement)
ZooKeeper Architecture

Distributed client-server application

ZooKeeper ensemble

Client

Follower

Leader

Follower

Client

sessions
ZooKeeper Architecture

- Ensemble elects a leader
- Writes complete after replicated to a strict majority
- Reads served from memory
ZooKeeper Scalability

- More servers => higher read load but lower update throughput
- *Observers* do not participate in agreement
  - Increase read capacity
  - Less impact on update
The Tail at Scale

- Problems are aggravated in large systems
  - Component-level *variability* amplified by scale
  - Failures and slow components are part of normal life, not an exception
  - Hard to keep tail of latency distribution short

**Goal**
Create a predictably responsive whole out of less-predictable parts
The Tail at Scale

Ways of addressing service variability

- **Prevent** bad things from happening by
  - Micro-partitioning, dynamic load balancing
  - Selective over-replication
  - Detecting and isolating the slow/flawed components

- **Contain** bad things through redundancy
  - Hedged/tied requests, speculative task execution
  - 99th-percentile: 1800ms -> 74ms, +2% requests
An Ideal System Should …

3. Be designed for the right goal
Expected Workload Matters

- Latency-oriented
  - Interactive, user-facing systems
  - Example: Web search serving

- Throughput-oriented
  - Back-end heavyweights
  - Example: Web search indexing
Access Patterns Matter

- **Data Analytics**
  - Throughput-oriented applications
  - Write-once (typically, append)
  - Read-many (typically, large sequential reads)

- **Online Transaction Processing (OLTP)**
  - Latency-oriented applications
  - Write-intensive
  - Typically, many small direct accesses

- Huge gray area in between
Locality Matters

- Can computation and storage be aligned?
  - Optimization?
- How repetitive is the workload?
  - Optimization?

\[ \Pr(x > X) \sim X^{-\alpha} \]
Consistency Matters

- Stricter properties = stronger consistency
  - Are you prepared to handle weird stuff?
- Stock alerts
  - Is it okay to lose an event once in a while?
- Social network
  - Bob deletes photos with his ex-date Alice
  - Bob befriends Carol
  - Can Carol observe these events in reverse order?
Example: Amazon’s Outage

9:06 AM PDT We are currently experiencing elevated error rates with S3. We are investigating.
9:27 AM PDT We're investigating an issue affecting requests. We will continue to post updates here.
9:48 AM PDT Just wanted to provide an update that we are currently pursuing several paths of corrective action.
10:13 AM PDT We are continuing to pursue corrective action.
10:33 AM PDT A quick update that we believe this is an issue with the communication between several Amazon S3 internal components. We do not have an ETA at this time but will continue to keep you updated.
11:02 AM PDT We're currently in the process of testing a potential solution.
11:23 AM PDT Testing is still in progress. We're working very hard to restore service to our customers.
11:45 AM PDT We're still in the process of testing a series of configuration changes aimed at bringing the service back online.
12:05 PM PDT We have now restored communication between a small subset of hosts. We are working on restoring internal communication across the rest of the fleet.
12:23 PM PDT We have restored communication between additional hosts and are continuing this work across the rest of the fleet. Thank you for your continued patience.
12:51 PM PDT The restored hosts are stable and we are moving forward in restoring communication between additional hosts.
1:17 PM PDT We continue to make incremental progress and communication between additional hosts has been restored. We are continuing with the plan to restore communication across Amazon S3’s large fleet of hosts.
1:39 PM PDT At this point, we are accelerating progress on restoring internal communication as all signs continue to look good.
2:03 PM PDT We have restored all internal communication between hosts in the EU and we are continuing to make progress in the US. Once all internal communication has been restored, we will start a multi-step process to begin accepting requests across Amazon S3 locations.
2:19 PM PDT A quick update to let you know that we have now also restored all internal communication between hosts in our West Coast facilities in the US.
2:36 PM PDT We have restored all internal communication across Amazon S3 hosts. We have started the multi-step process to begin accepting requests across Amazon S3 locations.
3:08 PM PDT We are attempting to bring EU back up now, followed by our US locations. EU will be first due to the smaller number of hosts. No data has been lost during this incident.

Weak consistency models can lead to data loss.
Summary

- Design for scale
- Design for fault-tolerance, tail-tolerance
- Know what you design for
Further Reading

- Designs, Lessons and Advice from Building Large Distributed Systems
- Lessons of Scale at Facebook
- Redesigning the Data Center (CACM)