HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Chapter 7:
Distributed Systems:
Warehouse-Scale Computing

Fall 2013
Jussi Kangasharju

Chapter Outline

- Warehouse-scale computing overview
- Workloads and software infrastructure
- Failures and repairs

- Note: Term "Warehouse-scale computing" originates from Google → Examples typically of Google's services
- Trend towards WSC is more general

- This chapter based on book Barroso, Hölzle: "The Datacenter as a Computer" (see course website)

What is Warehouse-Scale Computing (WSC)?

- Essentially: Modern Internet services

- Massive scale of…
  - Software infrastructure
  - Data repositories
  - Hardware platform

- Program is a service
- Consists of tens of interacting programs
  - Different teams, organizations, etc.

WSC vs. Data Centers

- Both look very similar to the outside
  - "Lots of computers in one building"

- Key difference:
- Data centers host services for multiple providers
  - Little commonality between hardware and software
  - Third-party software solutions

- WSC run by a single organization
  - Homogeneous hardware and software and management
  - In-house middleware

Cost Efficiency

- Cost efficiency extremely important

- Growth driven by 3 main factors:
    - Popularity increases load
    - Size of problem increases (e.g., indexing of Web)
    - Highly competitive market

- Need bigger and bigger systems → Cost efficiency!

Future of Distributed Computing?

- WSC is not just a collection of servers
  - New and rapidly evolving workloads
  - Too big to simulate → New design techniques
  - Fault behavior
  - Energy efficiency
  - New programming paradigms

- Design spectrum:
  - One computer → Multiple computers → Data center
  - WSC = Multiple data centers operating together
  - Modern CDN: "Server" = WSC data center

Architectural Overview

- Storage
- Networking
- Storage hierarchy
- Latency, bandwidth, capacity
- Power usage
- Handling failures

General architecture

- Servers, e.g., 1-U servers

- Racks

- Interconnected racks

Storage

- Tradeoff: NAS vs. local disks as distributed filesystem?

- NAS:
  - Easier to deploy, puts responsibility on vendor

- Collection of disks:
  - Must implement own filesystem abstraction (e.g., GFS)
  - Lower hardware costs (desktop vs. enterprise disks)
  - Reliability issues and replication?
  - More network traffic due to writes

Network

- 48-port 1 Gbps Ethernet switches are "cheap"

- Good bandwidth within one rack

- Problem: Cluster-level bandwidth?
  - Bigger and faster switches prohibitively expensive?

- Hierarchical network organization:
  - Good bandwidth within rack
  - Less bandwidth within cluster
  - Programmer must keep this in mind! (transparency?)

Storage Hierarchy

- Server:
    - N processors, X cores/CPU, local cache, DRAM, disks
    - Fast, but limited capacity
- Rack:
    - Individual servers, combined view
    - A bit slower, but more capacity
- Cluster:
    - View over all racks
    - Slower, but more capacity

- Tradeoff: Bandwidth, latency, capacity

Power Usage

- No single culprit on server level
  - CPU 33%
  - DRAM 30%
  - Disk 10%
  - Network 5%
  - Other 22%

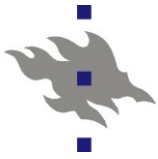- Further optimization targets on cluster/WSC level
  - Cooling of data center

Handling Failures

- At this scale, things will break often

- Application must handle them

- More details later

Workloads and Software Infrastructure

- Different levels of abstraction

- Platform-level software
  - Firmware, kernel, individual OS

- Cluster-level infrastructure software
  - Distributed software for managing resources and services
  - "OS for a datacenter"
  - Distributed FS, RPC, MapReduce, …

- Application-level software
  - Actual application, e.g., Gmail, Google Maps

Datacenter vs. Desktop

- Differences in developing software

- Datacenter:
    - Parallelism (both data and requests)
    - Workload changes
    - Homogeneous platform
    - Hiding failures

Basic Techniques

Technique	Reliability	Availability
Replication	Yes	Yes
Partitioning	Yes	Yes
Load balancing	Yes	
Timers		Yes
Integrity checks		Yes
App.-specific Compression	Yes	
Eventual consistency	Yes	Yes

Cluster-Level Infrastructure Software

- Resource management
  - Mapping of tasks to resources
- Hardware abstraction and basic services
  - Distributed storage, message passing, …
- Deployment and maintenance
  - Software distribution, configuration, …
- Programming frameworks
  - Hide some of the above from programmer
  - Examples: MapReduce, BigTable, Dynamo

MapReduce

- Google's framework for processing large data sets on clusters
- Name from map and reduce (functional programming)
  - Not really much in common with real "map" and "reduce"
- One master, multiple (levels) of slaves
- Map:
  - Master partitions input, distributed to slaves
  - Slaves may do the same
- Reduce:
  - Slave sends its result to its master
  - Eventually root-master will get result

Application-Level Software

- **What is the application?**
  - First was search, then many other have appeared

- **Datacenter must support general-purpose computing**
  - Too expensive to tailor datacenters for applications
  - Changing workloads → Faster to adapt software

- **Two application examples:**
  - Search
  - Similar scientific articles (see book for description)

Search

- Inverted index
    - Set of documents matching a keyword
- Size of index similar to original data
- Consider query "new york restaurant"
    - Must search each of three terms
    - Find documents matching every term
    - Sorting (PageRank + other criteria) → Result
- Latency must be low (user waiting)
- Throughput must be high (many users)
- Read-only index → Easily parallelizable

Monitoring Infrastructure

- Service-level dashboards
    - Real-time monitoring of few key indicators (latency, t-put)
    - Can extend to some more indicators
- Performance debugging tools
    - Dashboards only show problem, but no answer to "why"
    - No need for real-time (compare CPU profilers)
    - Blackbox monitoring vs. instrumentation approach
- Platform-level monitoring
    - Everything above is needed, but not sufficient
    - Need a higher-level view (see book for details)
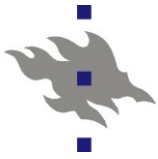
# Buy vs. Build?

- Buy:
  - Typical solution
- Build:
  - Google's (and others') approach
  - Original reason: No third-party solutions available
  - More software development and maintenance work
  - Improved flexibility
  - In-house software can take "shortcuts"
    - Not implement every feature

Failures

- Traditional software not good with failures

- Result: Make hardware more reliable

- WSC is different because of scale
  - Imaginary 30 year MTBF = 10,000 days MTBF
  - WSC with 10,000 servers = 1 failure per day

- Software must handle failures
  - Application or middleware
  - Middleware makes applications simpler
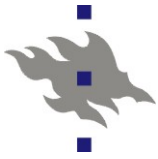
Positive Side Effect

- Failures are a fact of life

- Can buy cheaper hardware

- Upgrades are simpler
  - Upgrade, kill, reboot
  - Same for hardware upgrades

- "Failure is an option" ☺
  - Can allow servers to fail, makes life simpler

Caveats

- Cannot ignore reliability completely

- Hardware must be able to detect errors and failures
  - No need to recover, but can include

- Not detecting hardware errors is risky
  - See book for example
  - Every piece of software would need to handle everything

Categorizing Faults

- Corrupted
    - Data lost or corrupted
    - Can data be regenerated or not?
- Unreachable
    - Service unreachable by users
    - User network reliability?
- Degraded
    - Service available, but degraded
    - What can be still done?
- Masked
    - Fault occurs, but is masked

Sources of Faults

- Hardware not the common culprit (~10%)

- Software and configurations are bigger problems
    - Exact numbers depend on study

- Hardware problem = single computer
- Software/configuration problem = many computers simultaneously

Causes of Crashes

- Anecdotal evidence points to software
- Hardware: Memory or disk

- DRAM errors happen, but can be helped with ECC
  - Some errors still persist

- Real crash rate higher than studies predict
  - Again points to software

- Predicting problems in WSC not useful
  - Need to handle failures anyway
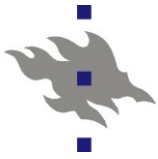 - Could be useful in other systems

Repairs

- When something breaks, it must be repaired

- Two important characteristics of WSC

- No need to repair immediately
  - Optimize time of repair technician

- Collect lot of health data from large number of servers
  - Use machine learning to optimize actions

Summary: Key Challenges

- Rapidly changing workloads

- Building balanced systems from imbalanced components

- Energy use

- Amdahl's Law

Chapter Summary

- Warehouse-scale computing overview

- Workloads and software infrastructure

- Failures and repairs