Warehouse-scale Data Centers

Kai Shen

Disclaimer

- Preparation of this class was helped by materials in the book "The Datacenter as a Computer" by Luiz A. Barroso, Jimmy Clidaras, and Urs Holzle.

Overview

- Very large data centers (size of warehouses)
  - Many machines, storage facilities, network interconnects, …
- Motivations for the extreme scale
  - Client/server computing (Internet services)
  - Highly popular applications /services
  - Big data
  - Cost efficiency at scale (buying bulk is cheaper)
    - Resources to applications can be elastically managed
    - Repair and maintenance is centralized
    - Mistakes from individual incidents are learned and used to improve management at scale
    - A source of big data to learn computer systems characteristics

Data Center Architecture

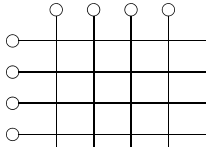
- Racks of machines, connected by networks
  - Figure 1.1 (page 6)
- Computer system scaling
  - Traditional supercomputers
    - custom network and node design
    - close integration but expensive, hard to scale
  - Massive cluster of commodity machines
    - commodity machines and networks, glued together by software
    - may scale better with the "right" software
    - cheaper at scale since the software cost is amortized

Storage Architecture

- Where to store the big data?
- Each machine has local storage, managed by distributed file system such as GFS (Colossus)
- Alternatively, storage area network
  - specialized storage systems plugged into the network switch that manages replication, parallel I/O etc. (scalable RAID system)
- Tradeoffs:
  - Intensity of writes in workload
  - Quality of service in I/O
  - Data locality through smart software management
    - Contrast of storage and network speeds
  - Complicated cost analysis (requirements on network capacity)

Network Fabric

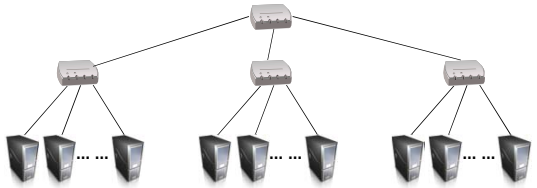
- Network matters to distributed data processing
- A Gbps network switch allows simultaneous traffic at full link speed
  - Crossbar: Complexity grows up quickly with the number of ports



  - Switch with a small number of ports is cheaper (~$30 per port)
  - Cost goes up non-linearly (an order of magnitude higher per port cost)
- One design choice
  - Cheap, few-port switch to link machines on a rack
  - Expensive, many-port switch to link all the racks together

Network Switching at Data Centers

- Thousands of servers with hierarchical network switching



- Reduce congestions:
  - Network load balance across nodes
  - Place highly communicating server modules within same switch domain, or better yet, in same machines
  - Redundant backbone links → Equal-Cost Multi-Path (ECMP) routing

Storage Hierarchy

- CPU cache, memory, and disk
  - Larger space with longer latency
- In a data center, we have CPU cache, local memory, local disk, rack memory, rack disk, data center memory, data center disk
  - Figures 1.3, 1.4 (page 10), 1.5 (page 11)

Data Center Workloads

- Unique, large jobs
  - E.g., compute page ranks from the 50B crawled web pages
- Many, small tasks
  - E.g., Amazon cloud computing, Google search service
- Parallelism
  - Data parallelism
  - User / request parallelism

Platform / Application Adaptation

- Platform: basic hardware and infrastructure system software
- Applications developed on top
- Platform / application adaptation
  - Platforms are somewhat homogeneous
  - Application upgrades are fast

  ⇨ Focus on basic cost-effectiveness on the platform construction and let application software adapt to the platform

Infrastructure Software: Buy, Adapt, Build

- Infrastructure software: operating system, file systems, databases, parallel computation, replication, …
- Choices:
  - Buy from vendors
  - Take open-source software and make adaptation
  - Build new software from scratch
- Google's argument: adapt or build, but don't buy
  - Need full control (performance debugging), flexibility, customization (nosql)
  - Few vendors (think of databases) are capable of building and testing their software at the datacenter scale

Cost-Efficient Analysis

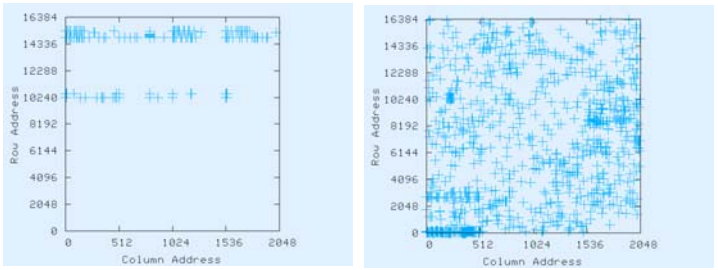
- You have N parallel tasks
  - c is the computation time, d is the number of data accesses, l_{local} is the latency of local data access, l_{remote} is the latency of remote data access
- If each task runs on a separate machine
  - Assume data is evenly distributed, the chance of a data access to be local is 1/N
  - Then the execution time is: $c + d \cdot (l_{local} \cdot 1/N + l_{remote} \cdot (1\text{-}1/N))$
- If you have a large multiprocessor (host all N tasks):
  - Then the execution time is: $c + d \cdot l_{local}$
- In practice, your multiprocessor isn't large enough to host all (host n<N tasks)
  - Then the execution time is: $c + d \cdot (l_{local} \cdot n/N + l_{remote} \cdot (1\text{-}n/N))$
  - Figure 3.2 (page 36)

Conventional vs. Wimpy CPUs

- Conventional CPU (regular CPUs on desktops / servers)
- Wimpy CPU (what you have in a smart phone, worse)
- Why wimpy CPUs?
  - With good parallelism, a larger number of wimpy CPUs can do the job done at a smaller number of conventional CPUs
  - Cluster of wimpy CPUs may save power and save money
- But the assumption of "good parallelism"
  - Now you know that good parallelism isn't easy to get

Reliability at Scale

- Chance of failures/anomalies to occur in a large system is high



- Permanent faults on two memory chips from thousands of memory chips we examined in a production system
- Transient faults were also found

Reliability at Scale

- Performance anomalies
  - A particular request / map task is very slow
- In a very large system, you may not always be able to find out problem cause and fix the problem, so
  - Detect problem and tolerate / mitigate it
  - Redundancy / replication
  - Technique to tolerate slow tail: perform work redundantly if the current run is too slow
  - Be prepared to recover by restarting and make sure that restart is fast
    - Maybe multi-scaled restart

Failure Correlation

- Correlated catastrophic failures
  - A bug-triggering input / request: Canaries
  - Load-triggered failures