Cloud Computing

- Cloud computing is a model for enabling on-demand network access to a shared pool of resources from a large infrastructure
- Resources
  - Storage
  - Computation
  - Applications
  - Services
  - Network bandwidth
- These resources can be rapidly provisioned and released with minimal management effort or service provider interaction

Why?

- Sharing resources among applications utilizes these resources better
  - While the resource needs of an application may vary dramatically, the sum of the resource needs of many independent applications varies much less
- Popularity of data-driven decision making
- Rapid prototyping and market testing
- No up-front cost by the users
  - Companies can start small (demand unknown in advance)
  - Increase resources only when there is an increase in need (elasticity)
  - 'Infinite' computing resources
  - Computation cost is proportional to use
    - "Cost associativity": 1 EC2 machine for 1000 hours = 1000 EC2 machines for 1 hour
  - Upfront setup cost is amortized over many applications
- Improved service levels, fault tolerance, and availability
- Better security infrastructure, better protection against network attacks
- Allows self-service deployment
- Costs continue to decrease every year
- Top paying technology

Current Trends

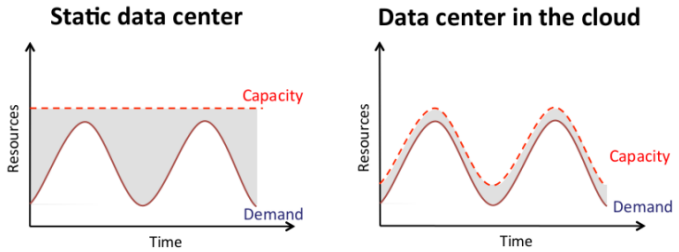
- Cloud providers: Amazon, Google, Microsoft, ...
- Public clouds: Amazon Web Services, Google Cloud Platform, Microsoft Azure, ...
  - They provide infrastructure and/or deployment platform
  - They offer them as web services and are billed on a utility basis
- Amazon web services
  - Simple Storage Service (SS3)
  - Elastic Compute Cloud (EC2)
  - Elastic MapReduce
- Google App Engine
  - Build your application in Node.js, Java, Ruby, C#, Go, Python, or PHP
  - A fully managed environment lets you focus on code while App Engine manages infrastructure concerns

Challenges

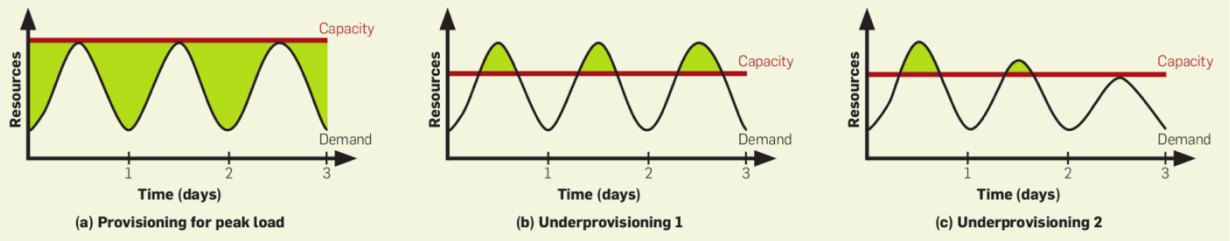
- Data security
  - A cloud is a high value target to hackers
- Have to depend on the security provided by the cloud host
- Data protection against other applications
- Protection against the cloud host
- Data ownership issues
- Moving sensitive data to the cloud
- Performance variability
- Data transfer bottlenecks

Economics of Cloud Providers

- Pay by use instead of provisioning for peak



  - Risk of over-provisioning: underutilization
  - Heavy penalty for under-provisioning



Resource	Cost in Medium Data Center	Cost in Very Large Data Center	Ratio
Network	$95/Mbps/month	$13/Mbps/month	7.1x
Storage	$2.20/GB/month	$0.40/GB/month	5.7x
Administration	≈140 servers/admin	>1000 servers/admin	7.1x

- Traditional in-house computing:
  - 5-7x more expensive than cloud computing
    - Although in some scenarios in-house computing can be cheaper
  - Power/cooling costs: approx. double cost of storage, CPU, network
- Added benefits (to cloud providers)
  - Utilize off-peak capacity (Amazon)
  - Sell .NET tools (Microsoft)
  - Reuse existing infrastructure (Google)
  - Cost per unit is decreasing

What is Big Data?

- The three Vs of big data: Volume, Variety, and Velocity
- High Volume (too big): petabyte-scale collections
- High Variety (too hard): data comes from a variety of sources in different formats
  - Irregular, unstructured, or incomplete data
  - Data need to be cleaned, processed, and integrated
- High Velocity (too fast): data need to be processed quickly (throughput) with low latency
  - Batch streaming, Real-time analytics
- Also, veracity: correctness and accuracy of data

Units of Data

Unit	Value	Scale
Kilobyte (KB)	1000 bytes	A paragraph of text
Megabyte (MB)	1000 KB	A short novel
Gigabyte (GB)	1000 GB	Beethoven's 5th symphony
Terabyte (TB)	1000 TB	All the x-rays in a large hospital
Petabyte (PB)	1000 PB	$\approx 1/2$ of the content of all US research libraries
Exabyte (EB)	1000 EB	$\approx 1/5$ of the words humans have ever spoken

How Big?

- Google
  - Processes 20 PB a day (2008)
  - Crawls 20B web pages a day (2012)
  - Search index is 100+ PB (5/2014)
  - Bigtable serves 2+ EB, 600M QPS (5/2014)
- Facebook
  - 300 PB data in Hive + 600 TB/day (4/2014)
- Amazon Web Services
  - S3: 2T objects, 1.1M request/ second (4/2013)
- Large Hadron Collider (LHC)
  - ~15 PB a year
- Square Kilometre Array (SKA)
  - 0.3 – 1.5 EB per year (~2020)

Challenges

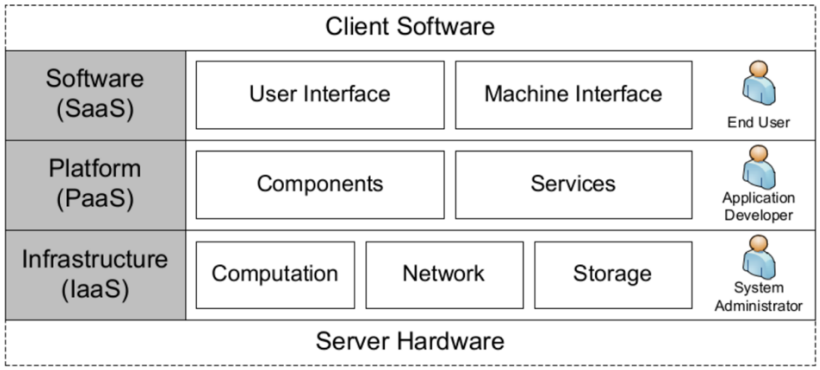
- The sources and amount of data keep growing
  - Many more companies try to collect data and turn them into value
    - Targeted advertising
  - E.g., social network data, Large Hadron Collider at CERN, Astronomy's Pan-STARRS5 array of celestial telescopes, Sloan Sky Survey, genome, climate data, oceanography, etc.
- Increased variety
  - logs, XML, JSON, etc.
- Data storage and processing can't keep up
  - A 1 TB hard disk costs ≈ $25 USD
  - Reading 1 TB from disk ≈ 5 hours

- So, it would cost $100,000 USD to store Facebook's 4PB of data/day and take ≈ 2.5 years to read!

Solution:

- Store the data on multiple machines
  - Added complexity of network communication
- Better virtualization technology
- Better networking devices

Service Offering Models



- Software as a Service
  - Applications running on a cloud infrastructure
  - Users: end users
  - E.g., some web applications, such as Gmail, Microsoft Office 365, etc.
- Platform as a Service
  - User-defined applications created using a set of provided libraries, services, and tools
  - Deploy onto the cloud infrastructure
  - Users: application developers
  - E.g., Google App Engine
- Infrastructure as a Service
  - Deploy and run arbitrary software, including operating systems and applications, share computing resources, storage, network
  - Users: IT architects, data analysts
  - E.g, Amazon Web Services (AWS)
- Database as a Service
  - Running a database on the cloud
  - Examples: Amazon RDS, Microsoft Azure SQL
- Search as a Service
  - Running a search engine on the cloud
  - Example: Amazon Elasticsearch Service
- Function as a Service
  - Run this function on the cloud
  - Example: Amazon Lambda, Google Cloud Functions

Scalability

a. Size scalability
   - Can easily add more users or resources to the system

b. Geographical scalability
   - Can easily handle users and resources that lie apart

c. Administrative scalability
   - Can easily manage a system that spans many independent administrative organizations

a. *Size Scalability*

Vertical Scaling: Ability to run large instances of a task

- Scale up/down
  - Scaling up means adding resources to a single node
  - Scaling down means removing resources from a single node
- Used in some scalable database systems

Horizontal scaling: Ability to run many instances of a task in parallel

- Scale in/out
  - Scaling out means adding more nodes to a system, such as adding a new computer to a cluster
  - Scaling in means removing nodes from a system
- Each node usually gets a modest amount of data
- Often with relaxed consistency
- Focus of Map-Reduce and many NoSQL systems

b. *Geographical Scalability*

- Challenges in scaling from LAN to WAN
  - Many distributed systems assume *synchronous* client-server interactions. E.g.
    - Client sends request and waits for an answer
    - Latency may easily prohibit this scheme
  - Large network latency in WAN
  - Building interactive application is non-trivial
  - Assumption of reliable communication
    - WAN is not reliable
    - E.g., locating a server through broadcasting is difficult

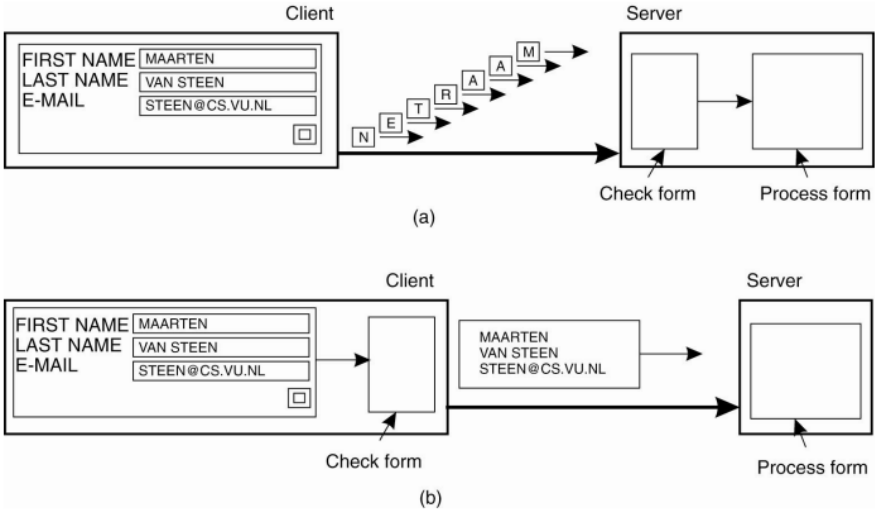
c. *Administrative Scalability*

- Conflicting policies with respect to
  - Resource usage and accounting
  - Management
  - Security

Latency

- Latency can be caused by:
  - Shared resources
  - Maintenance activities
    - E.g., log compaction, garbage collection
  - Queueing in intermediate servers
  - Power saving modes in modern processors
  - Stragglers
  - Large number of user requests
- Solutions:
  - Maintaining own shallow priority queue for asynchronous tasks
  - Break long running queues to smaller queues
  - Schedule maintenance services during lower overall load
  - Issue same request to multiple replicas
    - Causes high load; Send cancellation message to other servers after a request gets scheduled in one server
  - Divide the request into smaller tasks
  - Selective replication
    - Make more replicas of popular contents

- Move computations to the client



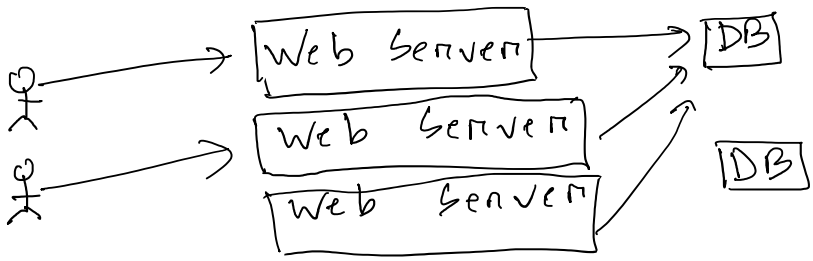
  - Use asynchronous communication
  - Move part of the computation to the client if applications can't use asynchronous communications efficiently

- Distribution: partition data and computations across multiple machines
  - Data parallelism in a multi-processor machine
    - Doesn't involve network communication
  - Data parallelism in a distributed setting
    - Introduces network latency
    - Sending a packet from US->Europe->US is 1,000,000x slower than reading from memory!

Numbers Everyone should know (- Jeff Dean)

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

The goal: scalable throughput
nx servers -> nx total throughput via parallel CPU, disk, net



- Buy a second computer (e.g., Web server) to help execute a problem so we can
  - Solve the problem in half the time or
  - Maybe solve twice as many problem instances
- So, handling more loads only requires buying more computers rather than re-design by expensive programmers
- Effective when you can divide work without much interaction
- Scaling gets harder as n grows:
  - Load imbalance, stragglers, slowest-of-n latency
  - Non-parallelizable code: initialization, interaction
  - Bottlenecks from shared resources, e.g., network
- Some performance problems aren't easily solved by scaling
  - E.g., quick response time for a single user request
  - E.g., all users want to update the same data
  - Often requires better design rather than just more computers
    - Replication:
      - Increases availability
        - Helps to balance the load, leading to better performance
      - Fault tolerance
        - 1000s of servers, big network -> always something broken
        - If one server crashes, can proceed using the other(s)

Why or why not High-performance computing system?
- Parallel or distributed computations on multiple disks and nodes
  - Often using message passing interface (MPI) and
  - Parallel virtual machines (PVM)
- Data are stored in a central location and copied to processing nodes when needed
- Brings the data to the computation
- Good for computationally intense tasks

Why not use a Traditional RDBMS?
- They are good for transactional data management
  - Banking, airline reservation, e-commerce, etc.
- Write-intensive, require ACID
  - Atomic: To the outside world, the transaction happens indivisibly
    - Abort otherwise
  - Consistent: The transaction does not violate system invariants
    - E.g., Credits must sum up to debits

    - Isolated: Concurrent transactions do not interfere with each other (Serializable)
        - The effects of all transactions appear to be one after another
    - Durable: once a transaction commits, the changes are permanent (written to disk/replicated)
- Do not typically use shared-nothing architecture
- Hard to maintain ACID guarantees in the face of data replication over WAN
- Today's SQL databases cannot scale to the thousands of nodes deployed in the cloud context
- Hard to support multiple, distributed updates to the same data set
- Hard to replicate huge data sets for availability
- Full ACID guarantees are typically not required in analytical applications
- Seek time is improving more slowly than transfer rate
    - Seeking is the process of moving the disk's head to a particular place on the disk to read or write data
    - It characterizes the latency of a disk operation, whereas the transfer rate corresponds to a disk's bandwidth
- An RDBMS is good for point queries or updates
    - Where the dataset has been indexed to deliver low-latency retrieval and update times of a relatively small amount of data
- MapReduce suits applications where the data is written once and read many times
    - Whereas a relational database is good for datasets that are continually updated

NoSQL Systems

- Stands for: Not Only SQL
- No schema
- Simple and scalable data management systems
- Typically oriented towards clusters and cloud (but not all)
- No "one size fits all" philosophy
- Many are open source
- Various types:
    - Key-value stores
    - Document stores
    - Extensible record stores
- Databases vs NoSQL:
    - Database: complex/concurrent
    - NoSQL: simple/scalable
    - Transactional vs analytical

Big Data: New Technologies

- Big Data Processing System
    - Run on clusters of share-nothing computers
    - Often using cloud computing
    - Use distributed storage
        - The data are already distributed when they are stored
    - Distributed computation: run computations where the data is
        - Brings the computation to the data
    - Good for data intense tasks
- Infrastructure

    - New computing paradigms: MapReduce, Hadoop, Spark
    - New storage solutions: HDFS
    - New languages: Hive, Pig Latin
- Algorithms and techniques
    - New infrastructure demands new approaches to explore data
    - We will study algorithms to process and explore data in Big-Data environments

Wish List

- Scalability
    - Data analysis is elastic, but only if workload is *parallelizable*
        - No need to rewrite everything for a distributed system
    - Scale out, not up
        - Symmetric multiprocessing (SMP) and large shared-memory systems don't scale well
- Fault tolerance
    - Fault tolerant read-only query: does not have to restart it if one of the processing nodes fails
        - Complex queries can involve thousands of nodes and can take hours to complete
        - If the cloud service is built on top of cheap, commodity hardware failure is not uncommon
        - The probability of a failure occurring during a long-running data analysis task is relatively high
        - E.g., One server may stay up 3 years (approx. 1000 days)
            - For 1000 servers: 1 fail per day
            - Google had 1M machines in 2011: 1000 fails per day!
- Data locality
    - Move the computation to the data
- Process data sequentially, avoid random access
    - Seeks are expensive, disk throughput is good
    - Seek vs Scan
        - Consider a 1 TB database with 100 byte records
        - We want to update 1 percent of the records
        - Option 1: Mutate each record
            - Each update takes ~30 ms (seek, read, write)
            - 10^8 updates = ~35 days
        - Option 2: Rewrite all records
            - Assume 100 MB/s throughput
            - Time = 5.6 hours(!)
        - Random access is expensive

Optional Read:

- More data usually beats better algorithms, by Ananad Rajaraman
- The Unreasonable Effectiveness of Data, by Alon Halevy, Peter Norvig, and Fernando Pereira, IEEE Intelligent Systems, March/April 2009
    - https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35179.pdf