Cloud Data Management

Big Data

Vera Goebel
Fall 2015
Cloud Computing

• The vision
  – On demand, reliable services provided over the Internet (the “cloud”) with easy access to virtually infinite computing, storage and networking resources

• Simple and effective!
  – Through simple Web interfaces, users can outsource complex tasks
    • Data mgmt, system administration, application deployment
  – The complexity of managing the infrastructure gets shifted from the users' organization to the cloud provider

• Capitalizes on previous computing models
  – Web services, utility computing, cluster computing, virtualization, grid computing
Cloud Definition

• Def. 1: A cloud provides on demand resources and services over the Internet, usually at the scale and with the reliability of a data center.
• Def. 2: A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
• Everything gets delivered as a service:
  – Pay-as-you-go pricing model, whereby users only pay for the resources they consume
  – Service Level Agreement (SLA) to govern the use of services by customers and support pricing
    • E.g. the service uptime during a billing cycle (e.g. a month) should be at least 99%, and if the commitment is not met, the customer should get a service credit
Cloud Characteristics (1)

• On-demand service
• Broad network access
• Resource pooling
• Rapid elasticity
• Measured Service
Cloud Characteristics (2)

- Compute power is **elastic**, but only if workload is parallelizable -> applications must be designed to run on shared-nothing architecture
- Data is stored at **untrusted** hosts!
- Data is **replicated**, often across large geographic distances -> transactions?, consistency?
Cloud Service Models

- **Software as a Service (SaaS)**
  - Salesforce.com
  - Google App (Gmail, Docs,...)

- **Platform as a Service (PaaS)**
  - Google AppEngine
  - Microsoft Azure
  - Amazon RDS

- **Infrastructure as a Service (IaaS)**
  - Amazon EC2
  - Amazon S3
  - Rackspace
Cloud Taxonomy

• Infrastructure-as-a-Service (IaaS)
  • Computing, networking and storage resources, as a service
  • Provides elasticity: ability to scale up (add more resources) or scale down (release resources) as needed
  – E.g. Amazon Web Services

• Software-as-a-Service (SaaS)
  – Application software as a service
  – Generalizes the earlier ASP model with tools to integrate other applications, e.g. developed by the customer (using the cloud platform)
  – Hosted applications: from simple (email, calendar) to complex (CRM, data analysis or social network)
  – E.g. Salesforce CRM system

• Platform-as-a-Service (PaaS)
  – Computing platform with development tools and APIs as a service
  – Enables developers to create and deploy custom applications directly on the cloud infrastructure and integrate them with applications provided as SaaS
  – Ex. Google Apps
Cloud Deployment Models

- Private cloud
- Community cloud
- Public cloud
- Hybrid cloud
Related Technologies

- Virtualization
- Grid Computing
- Utility Computing
- Autonomic Computing
Virtualization & Cloud Computing
Grid Architecture

- Access through Web services to distributed, heterogeneous resources
  - supercomputers, clusters, databases, etc.
- For Virtual Organizations
  - which share the same resources, with common rules and access rights
- Grid middleware
  - security, database, provisioning, job scheduling, workflow management, etc.
Cloud Architecture

- Like grid, access to resources using Web services
  - But less distribution, more homogeneity, and bigger clusters
- For different customers
  - Including individuals
- Replication across sites for high availability
- Scalability, SLA, accounting and pricing essential

Key Components:
- User 1
  - Create VMs
  - Start VMs
  - Terminate VMs
  - Pay
- User 2
  - Reserve VMs
  - Store
  - Pay

WS calls

Clusters:
- Cluster 1
  - Service nodes
  - Compute nodes
  - Storage nodes
- Cluster 2
  - Service nodes
  - Compute nodes
  - Storage nodes
Evolution Towards Cloud Computing
Cloud Benefits

• Reduced cost
  – Customer side: the IT infrastructure needs not be owned and managed, and billed only based on resource consumption
  – Cloud provider side: by sharing costs for multiple customers, reduces its cost of ownership and operation to the minimum

• Ease of access and use
  – Customers can have access to IT services anytime, from anywhere with an Internet connection

• Quality of Service (QoS)
  – The operation of the IT infrastructure by a specialized, experienced provider (including with its own infrastructure) increases QoS

• Elasticity
  – Easy for customers to deal with sudden increases in loads by simply creating more virtual machines (VMs)
Main Issue: Security and Privacy

- Current solutions
  - **Internal cloud (or private cloud)**: the use of cloud technologies but in a private network behind a firewall
    - Much tighter security
    - Reduced cost advantage because the infrastructure is not shared with other customers (as in public cloud)
    - Compromise: **hybrid cloud** (internal cloud for OLTP + public cloud for OLAP)
  - **Virtual private cloud**: Virtual Private Network (VPN) within a public cloud with security services
    - Promise of a similar level of security as an internal cloud and tighter integration with internal cloud security
    - *But such security integration is complex and requires talented security administrators*
Data Management in the Cloud

• Data management applications are potential candidates for deployment in the cloud
  – **industry**: enterprise database system have significant up-front cost that includes both hardware and software costs
  – **academia**: manage, process and share mass-produced data in the cloud

• Many “Cloud Killer Apps” are in fact data-intensive
  – Batch Processing as with map/reduce
  – Online Transaction Processing (OLTP) as in automated business applications
  – Offline Analytical Processing (OLAP) as in data mining or machine learning
Cloud Data Management

• Goals:
  – Availability
  – Scalability
  – Elasticity
  – Performance
  – Multitenancy
  – Load and tenant balancing
  – Fault tolerance
  – Ability to run in a heterogeneous environment
  – Flexible query interface

• Challenges:
  – Availability of a Service
  – Data Confidentiality
  – Data lock-in
  – Data transfer bottlenecks
  – Application parallelization
  – Shared-nothing architecture
  – Performance unpredictability
  – Application debugging in large-scale distributed systems
Data Management in the Cloud

• Two largest components of data management market:
  – Transactional Data Management
  – Analytical Data Management

• Which one will benefit from moving to the cloud?
Transactional Data Management

• Banks, airline reservation, online e-commerce
• ACID, write-intensive
• Not ready to move to the cloud for the following reasons:
  – Don’t use shared-nothing architecture
  – Hard to maintain ACID when data replication are all over the world
  – Enormous risks in storing transactional data on an untrusted host
Analytical Data Management

• Business planning, decision support
• Well-suited to run in a cloud environment:
  – Shared-nothing architecture is a good match
  – ACID guarantees are typically not needed
  – Particularly sensitive data can be left out of the cloud.
• **OLTP**
  - Operational databases of average sizes (TB), write-intensive
  - ACID transactional properties, strong data protection, response time guarantees

• **OLAP**
  - Historical databases of very large sizes (PB), read-intensive, can accept relaxed ACID properties

• **Not very suitable for cloud**
  - Requires shared-disk multiprocessors
  - Corporate data gets stored at untrusted host

• **Suitable for cloud**
  - Shared-nothing clusters of commodity servers are cost-effective
  - Sensitive data can be hidden (anonymized) in the cloud
Why not Relational DBS?

• Rel. DBS all have a distributed and parallel version
  – With SQL support for all kinds of data (structured, XML, multimedia, streams, etc.)

• But the “one size fits all” approach has reached the limits
  – Loss of performance, simplicity and flexibility for applications with specific, tight requirements
  – New specialized DBMS engines better: column-oriented DBMS for OLAP, DSMS for stream processing, etc.

• For the cloud, Rel. DBS provide both
  – Too much: ACID transactions, complex query language, lots of tuning knobs
  – Too little: specific optimizations for OLAP, flexible programming model, flexible schema, scalability
Cloud Data Management Solutions

• Cloud data
  – Can be very large (e.g. text-based or scientific applications), unstructured or semi-structured, and typically append-only (with rare updates)

• Cloud users and application developers
  – In very high numbers, with very diverse expertise but very little DBMS expertise

• Therefore, current cloud data management solutions trade consistency for scalability, simplicity and flexibility
  – New file systems: GFS, HDFS, …
  – New DBMS: Amazon SimpleDB, Google Base, Google Bigtable, Yahoo Pnuts, etc.
  – New parallel programming: Google MapReduce (and its many variations)
Design Decision of Cloud Storage Systems

- **CAP Theorem** - a shared-data system can only choose at most 2 out of 3 properties:
  - Consistency
  - Availability
  - Tolerance to Partitions
NoSQL

Typical traits

● Avoid join operations
● Scale horizontally
● No ACID guarantees

Other NoSQL Systems

● Apache's HBase
  ● Modeled after Bigtable
● Apache's Cassandra
  ● Facebook
  ● Digg
  ● Reddit
  ● Rackspace (ISP)
  ● Cloudkick
● LinkedIn's Project Voldemort
● Amazon's Dynamo
Cloud DBMS Wish List

- Efficiency
- Fault Tolerance
- Ability to run in a heterogeneous environment
- Ability to operate on encrypted data
- Ability to interface with business intelligence applications
MapReduce-like Software

- **Fault Tolerance**
  - MapReduce is designed with high fault tolerance support
- **Ability to run in a heterogeneous environment**
  - MapReduce is also carefully designed to run in a heterogeneous environment
- **Ability to operate on encrypted data**
  - No
- **Ability to interface with business intelligence applications**
  - No
- **Efficiency**
  - An order of magnitude slower than alternative systems
  - Not initially designed to work as a data analysis system
  - The performance is dependent on the application.
Shared-Nothing Parallel Databases

- Efficiency
  - Good with data analysis
- Fault Tolerance
  - Restart queries on failure
  - Failures are rare
- Ability to run in a heterogeneous environment
  - Poor
  - But not its fault
- Ability to operate on encrypted data
  - Poor: moving, copying only
  - Some support user defined functions
- Ability to interface with business intelligence applications
  - Good: JDBC, ODBC
A Call for a Hybrid Solution

• Combine MapReduce with shared-nothing DBMS to produce a system that better fit the cloud computing market

• Several aspects need to be balanced:
  – fault tolerance and performance
  – SQL on MR or/and reusable functions on SN DB
Google Technology - Motivation

- Nothing is small in Google land
  - Peta-bytes of data
  - Millions of users
  - Lots of services and servers
→ **Scalability**

- Failures are normal
  - Network connections
  - Hard disks
  - Power supplies
→ **Fault tolerance**

- Monitoring and maintenance is hard
→ **Autonomic computing**

- Clusters all over the world
- Thousands of queries served per second
  - One query reads hundreds of MB of data
  - One query consumes billions of CPU cycles

- A distributed, fault-tolerant file system is needed!
Google Data Centers

- Scaling out on commodity hardware is cheaper than scaling up on high-end servers
- Google servers:
  - > 15,000 servers (2003)
  - ~ 200,000 (2005)
  - ~ 1 M servers (2010)
- Data centers are composed of standard shipping containers with 1160 servers in each
Google Data Centers
Google File System (GFS)

• Used by many Google applications
  – Search engine, Bigtable, Mapreduce, etc.
• The basis for popular Open Source implementations: Hadoop HDFS (Apache & Yahoo)
• Optimized for specific needs
  – Shared-nothing cluster of thousand nodes, built from inexpensive hardware => node failure is the norm!
  – Very large files, of typically several GB, containing many objects such as web documents
  – Mostly read and append (random updates are rare)
    • Large reads of bulk data (e.g. 1 MB) and small random reads (e.g. 1 KB)
    • Append operations are also large and there may be many concurrent clients that append the same file
    • High throughput (for bulk data) more important than low latency
GFS: Design Choices

- Traditional file system interface (create, open, read, write, close, and delete file)
  - Two additional operations: snapshot and record append.
- Relaxed consistency, with atomic record append
  - No need for distributed lock management
  - Up to the application to use techniques such as checkpointing and writing self-validating records
- Single GFS master
  - Maintains file metadata such as namespace, access control information, and data placement information
  - Simple, lightly loaded, fault-tolerant
- Fast recovery and replication strategies
GFS: Distributed Architecture

- Files are divided in fixed-size partitions, called *chunks*, of large size, i.e. 64 MB, each replicated at several nodes.
Chunks & Chunk Servers

**Chunk**

- Similar to block in file systems
- Size is always 64 MB
- Less fragmentation
- Eases management
- Sent directly to clients
Master Servers

**Master Server**

- Coordinates cluster
- Updates operation log
- Stores meta-data
Master Server – Chunk Server Communication

**State updates**
- Is a chunk server down?
- Are there disk failures on a chunk server?
- Are any replicas corrupted?
- Which chunk replicas does a chunk server store?

**Instructions**
- Create new chunk
- Delete existing chunks
Google Bigtable

• Database storage system for a shared-nothing cluster
  – Uses GFS to store structured data, with fault-tolerance and availability

• Used by popular Google applications
  – Google Earth, Google Analytics, Orkut, etc.

• The basis for popular Open Source implementations
  – Hadoop Hbase on top of HDFS (Apache & Yahoo)

• Specific data model that combines aspects of row-store and column-store DBMS
  – Rows with multi-valued, timestamped attributes
    • A Bigtable is defined as a multidimensional map, indexed by a row key, a column key and a timestamp, each cell of the map being a single value (a string)

• Dynamic partitioning of tables for scalability
Bigtable

- Similar to a database, but not a full relational data model
- Data is indexed using row and column names
- Treats data as uninterpreted strings
- Clients can control the locality of their data
- Built on GFS
- Uses MapReduce
- Used by over 60 Google products
Servers

- Mapreduce worker
- Application server
- Bigtable server
- GFS server

Cluster management system

Linux
Data Model

- A cluster is a set of machines with Bigtable processes
  - Each Bigtable cluster serves a set of tables
- A **table** is a sparse, distributed, persistent multidimensional sorted map
- The data in the tables is organized into three dimensions:
  - Rows, Columns, Timestamps
  - \( \text{row:} \text{string, column:} \text{string, time:} \text{int64} \) \( \rightarrow \) string
- A **cell** is the storage referenced by a particular row key, column key and timestamp
Example Slice of a Table
Rows

- Bigtable maintains data in alphabetical order by row key
- The row keys in a table are arbitrary strings
- Rows are the unit of transactional consistency
- Several rows are grouped in **tablets** which are distributed and stored close to each other
  - Reads of short row ranges are efficient, typically require communication with only one or a few machines
  - Remember the reversed URIs in the example?
Columns

- Column keys are grouped into **column families**
- A column key is named with syntax → family:qualifier
- Data stored in a column family is usually of the same type

- An example column family for our previous example is language, which stores the language in which a web page was written.
Timestamps

- A cell can hold multiple versions of the data
- Timestamps can be set by Bigtable or client applications
- Data is stored so that new data are fastest to read
- Garbage-collection
  - Based on items (last x versions)
  - Based on time (last seven days)
- In the example, three versions of a web page was kept
Application Programming Interface

- Create and delete tables and column families
- Change cluster, table, and column family metadata (such as access control rights)
- Write or delete values
- Look up values from individual rows
- Iterate over a subset of the data in a table
- Data transformation (filtering)
- I/O of MapReduce jobs
A Bigtable Row

<table>
<thead>
<tr>
<th>Row unique id</th>
<th>Column family</th>
<th>Column key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row key</td>
<td>Contents:</td>
<td>Anchor:</td>
</tr>
<tr>
<td>&quot;com.google.www&quot;</td>
<td>&quot;&lt;html&gt; ...&lt;/html&gt;&quot;</td>
<td>&quot;inria.fr&quot; &quot;google.com&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;&lt;html&gt; ...&lt;/html&gt;&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Column family = a kind of multi-valued attribute
- Set of columns (of the same type), each identified by a key
  - Column key = attribute value, but used as a name
- Unit of access control and compression
Bigtable DDL and DML

• Basic API for defining and manipulating tables, within a programming language such as C++
  – Various operators to write and update values, and to iterate over subsets of data, produced by a scan operator
  – Various ways to restrict the rows, columns and timestamps produced by a scan, as in relational select, but no complex operator such as join or union
  – Transactional atomicity for single row updates only
Dynamic Range Partitioning

• Range partitioning of a table on the row key
  – Tablet = a partition corresponding to a row range.
  – Partitioning is dynamic, starting with one tablet (the entire table range) which is subsequently split into multiple tablets as the table grows
  – Metadata table itself partitioned in metadata tablets, with a single root tablet stored at a master server, similar to GFS’s master

• Implementation techniques
  – Compression of column families
  – Grouping of column families with high locality of access
  – Aggressive caching of metadata information by clients
Yahoo! PNUTS

- Parallel and distributed database system
- Designed for serving Web applications
  - No need for complex queries
  - Need for good response time, scalability and high availability
  - Relaxed consistency guarantees for replicated data
- Used internally at Yahoo!
  - User database, social networks, content metadata management and shopping listings management apps
Design Choices

• Basic relational data model
  – Tables of flat records, Blob attributes
  – Flexible schemas
    • New attributes can be added at any time even though the table is being queried or updated
    • Records need not have values for all attributes

• Simple query language
  – Selection and projection on a single relation
  – Updates and deletes must specify the primary key

• Range partitioning or hashing of tables into tablets
  – Placement in a cluster (at a site)
  – Sites in different geographical regions maintain a complete copy of the system and of each table

• Publish/subscribe mechanism with guaranteed delivery, for both reliability and replication
  – Used to replay lost updates, thus avoiding a traditional database log
Relaxed Consistency Model

• Between strong consistency and eventual consistency
  – Motivated by the fact that Web applications typically manipulate only one record at a time, but different records may be used under different geographic locations

• Per-record timeline consistency: guarantees that all replicas of a given record apply all updates to the record in the same order

• Several API operations with different guarantees
  – Read-any: returns a possibly stale version of the record
  – Read-latest: returns the latest copy of the record
  – Write: performs a single atomic write operation
MapReduce

- For data analysis of very large data sets
  - Highly dynamic, irregular, schemaless, etc.
  - SQL or Xquery too heavy

- New, simple parallel programming model
  - Data structured as (key, value) pairs
    - E.g. (doc-id, content), (word, count), etc.
  - Functional programming style with two functions to be given:
    - Map(k1,v1) $\rightarrow$ list(k2,v2)
    - Reduce(k2, list (v2)) $\rightarrow$ list(v3)

- Implemented on GFS on very large clusters
Fault-tolerance

• Fault-tolerance is fine-grain and well suited for large jobs

• Input and output data are stored in GFS
  – Already provides high fault-tolerance

• All intermediate data is written to disk
  – Helps checkpointing Map operations, and thus provides tolerance from soft failures

• If one Map node or Reduce node fails during execution (hard failure)
  – The tasks are made eligible by the master for scheduling onto other nodes
  – It may also be necessary to re-execute completed Map tasks, since the input data on the failed node disk is inaccessible
Issues in Cloud Data Mgmt.

- Main challenge: provide ease of programming, consistency, scalability and elasticity at the same time, over cloud data.
- Current solutions
  - Quite successful for specific, relatively simple applications
  - Have sacrificed consistency and ease of programming for the sake of scalability
  - Force applications to access data partitions individually, with a loss of consistency guarantees across data partitions
- For more complex apps. with tighter consistency requirements
  - Developers are faced with a very difficult problem: providing isolation and atomicity across data partitions through careful engineering
Future Trends

• The **Data Age!** -> Zettabytes of data (++)
• Mobile interactive applications
• Parallel batch processing
• Rise of analytical applications
• Backend-support for compute-intensive desktop applications
Research Directions

• Declarative programming languages for the cloud
  – E.g. BOOM project (UC Berkeley) using Overlog
• Parallel OLAP query processing with consistency guarantees wrt concurrent updates
  – E.g. using snapshot isolation
• Scientific workflow management
  – E.g. with P2P worker nodes
• Data privacy preserving query processing
  – E.g. queries on encrypted data
• Autonomic data management
  – E.g. automatic management of replication to deal with load changes
• Green data management
  – E.g. optimizing for energy efficiency