Managing Data in Cloud

File system

- Model to organize data into files and directories
- Accessed by attaching a virtual disk to a virtual machine

Blob Storage

- Binary Large Object
- Flat object model for data
- Extremely scalable

Databases

- Highly structured data collections
- Three types of databases
  1. Relational databases – Based on relational algebra and accessed by SQL
  2. Tables and NoSQL databases – easily distributed over multiple machines
  3. Graph databases – data represented by graphs (nodes/edges)

Storage as a Service

Three motivating examples

- Cloud services as data services
  - Run in cloud, host digital content in cloud, provide apps to access, store, and share that content
- Three use cases
  1. Set of simulation output files in climate science lab
     - Network Common Data Form (NetCDF) format
     - 20TB of data
     - Accessibility via interactive tools through web portal
     - Data to be partitioned to enable distributed analysis over multiple machines in parallel
  2. Records of experimental observations in seismic observatory
     - Time of observation, experimental parameters, and measurement in CSV format
     - A million records of 100 TB
     - Data stored for easy access by a large team and to permit tracking of data inventory and accesses
  3. Team of scientists operating a collection of several thousand instruments
     - Each instrument generates a small data record every few seconds
     - Analysis across entire collection every few hours poses data management challenge
     - Similar to analyzing large web traffic or social media stream

Storage models

- Support for many different storage models in cloud
  - Highly scalable from MB to hundreds of TB

- File systems
  - Tree of directories
  - Standard API of Unix-derived version of file system is Portable Operating System Interface (POSIX)
    - Read/write/delete files within directories
  - Allows for direct use of many existing programs without modification
  - Most people are familiar with it (navigation, code development, file sharing via email)
  - Straightforward mechanism to represent hierarchical relationship among data
  - Concurrent access by multiple readers
  - Disadvantages
    - No support to enforce conventions concerning representation of data elements and their relationship
    - No support to help user navigate complex data collections
    - Problem with scalability: need to maintain consistency as multiple processes read/write a file system, leading to bottlenecks in file system implementation
- Object stores
  - Stores unstructured binary objects
  - Blobs, or Binary Large Objects
  - Object stores eliminate hierarchy and forbids updates to objects once created
  - Two-level folder-file hierarchy, creating object containers
    - Each object container can hold zero or more objects
    - Each object identified by a unique id and can have various metadata associated with it
    - Objects cannot be modified once uploaded; they can be deleted or replaced
  - Advantages
    - Simplicity, performance, reliability
    - Since objects cannot be modified, you can create highly scalable and reliable implementations
      - Replicate an object across multiple storage devices to increase resilience and performance
      - No need for synchronization logic for concurrent updates
    - Objects can be moved manually or automatically among storage classes with different performance/cost parameters
  - Limitations
    - Little support to organize data
    - No support for search: object can be accessed only by its id
    - Need a separate index to map from object characteristics to its id
    - No mechanism to work with structured data
    - Object store cannot be mounted as a file system or accessed with existing tools in the ways file system can
- Relational databases
  - Database: an organized collection of data
  - Models real world objects in the form of entities and relationships
  - DBMS safely stores and efficiently manages databases, and allows discovery of relationships between entities
  - Three components: data model, query language, transactions/crash recovery
  - Advantages
    - Simplified data management and manipulation
    - Efficient querying and analysis

      * Durable and reliable storage
      * Scaling to large data size
      * Validation of data formats
      * Management of concurrent accesses
  - Properties
      * Tabular data, rows represent entities and columns represent attributes
      * Uses SQL to specify a range of powerful operations on tables
      * Sophisticated indexing and query planning techniques
      * Based on relational algebra, giving efficient and correct implementations
      * ACID semantics
          · Atomiticity – Entire transaction succeeds or fails
          · Consistency – Data collection never left in invalid or conflicting state
          · Isolation – Concurrent transactions cannot interfere with each other
          · Durability – After completion, system failures cannot invalidate the result

- NoSQL databases
  - Suitable for data that may not be rigidly structured (text)
  - Scale the quantities of data and number of users that can be supported
  - Key-value store
      * Organize large number of records
      * Each record associates an arbitrary key with an arbitrary value
  - Document store
      * A variant of key-value store that permits text search on the stored values
  - Limitations
      * Do not support full relational algebra
      * Do not support queries that join two tables
  - "Not only SQL"
      * May allow for rapid accumulation of unstructured data
      * Arbitrary data can be stored without modification to a DB schema; new columns introduced over time as data/understanding evolves
  - May not satisfy ACID semantics
      * Databases may be distributed over multiple servers and replicated over multiple data centers
      * *Consistency* may be replaced by *eventual consistency*; DB state may be momentarily inconsistent across replicas
  - The CAP theorem
      * Consistency: All computers see the same data at the same time
      * Availability: Every request receives a response about whether it succeeded or failed
      * Partition tolerance: System continues to operate even if a network failure prevents computers from communicating

        Theorem 1 *It is not possible to create a distributed system with all three properties.*
      * Creates a challenge with large transactional datasets
      * Distribution needed for high performance but large number of computers leads to likelihood of network disruption
      * Strict consistency cannot be achieved
      * DB designer must chooses between high consistency or high availability

    * Choose high availability for checkout in e-commerce setting; errors hidden from the user and handled later when adding items to a shopping cart
    * For final order submission, favor consistency because several services (credit card processing, shipping and handling, reporting) need to access data simultaneously

- Graph databases
  - $G = (\{V\}, \{E\})$
  - Search data based on relationships among data items
  - Often built on top of existing NoSQL databases
- Data warehouses
  - Data management systems optimized to support analytic queries that involve reading large datasets

Cloud storage landscape

- File systems
  - Also known as *file shares*
  - Virtual data drives that can be attached to virtual machines
  - Amazon's *Elastic Block Store* (EBS)
    * Device to be mounted onto a single EC2 compute server instance at a time
    * Low-latency access to data from a single EC2 instance
    * Working data that is read/written frequently by an application but too large to fit into memory
    * Accessible only to EC2 instances (inside Amazon cloud)
  - Amazon's *Elastic File System* (EFS)
    * General-purpose file storage service
    * Provides file system interface, file system access semantics (strong consistency, file locking), and concurrently-acessible storage for many EC2 instances
    * EFS can hold state that is to be read/written by many concurrent processes
    * Accessible only to EC2 instances (inside Amazon cloud)
  - Google Compute Engine
    * Offers three types of attached disks, and a way to attach an object store
      1. Persistent disk – up to 64 TB in size, most inexpensive, accessible anywhere in a zone
      2. Local SSD – Higher performance, more expensive, up to 3 TB, accessible only in the instance to which attached
      3. RAM disk – In-memory, up to 208 GB, expensive, accessible only in the instance to which attached
  - Azure File Storage
    * Allows creation of file shares accessible by a special protocol SMB
    * SMB lets Windows and Linux VMs to mount file shares natively
    * The file shares can be mounted on a user's Windows or Mac
- Object stores
  - Amazon's Simple Storage Service (S3)
    * Uses containers called *buckets* to hold objects
    * A related service – Glacier – provides long-term, secure, durable, and extremely low cost data archiving
    * Glacier access time may be several hours making it unsuitable for applications needing rapid data access

- Google's Cloud Storage
  - Basic object storage; durable, replicated, and highly available
  - Supports three storage tiers, with different performance and pricing levels
    1. Standard – Most expensive, multiregional; used for data that is accessed often
    2. Regional – Mid-range; used for batch jobs with noncritical response time
    3. Nearline – Inexpensive; cold storage and disaster recovery
  - There is also Coldline similar to Amazon's Glacier
- Azure Storage
  - Azure Storage explorer tool `storageexplorer.com` to see and manage services on Windows, Mac, and Linux
  - Use Azure Blob storage service for highly reliable storage of unstructured objects
  - Provides tiered storage and pricing given by *hot* and *cool*

- NoSQL services
  - Amazon's DynamoDB
    - Based on extended key-value model
    - The only required attribute is the primary key
    - Any number of additional columns may be defined, indexed, and made searchable in multiple ways
  - Amazon Elastic MapReduce (EMR)
    - Allows analysis of large quantities of data with Spark and other data analytics platforms
  - Google Cloud Bigtable
    - Powers many Google services including search, analytics, maps, and gmail
    - Maps two arbitrary strings (row key and column key) and a timestamp to an associated arbitrary byte array
      - Timestamp helps with versioning and garbage collection
    - Provides low latency and high bandwidth; efficient in terms of space and for massive workloads
    - Deployed on a Google-hosted dynamically-resizable cluster
  - Google Cloud Datastore
    - Similar to Google Bigtable
    - Implements ACID semantics
    - Rich set of SQL-like operators
  - Azure Table Storage
    - Simple NoSQL key-value store
    - Supports highly reliable storage of large amounts of data
    - Limited query capabilities with modest cost
  - Azure HDInsight
    - Hadoop storage service
    - Implements popular big data tools – Spark, HBase NoSQL database, Hive SQL database
  - DocumentDB
    - Similar to Table
    - Supports full text indexing and query, but at a higher cost

- Relational databases
  - Mature technology with deployments that scale to especially large size
  - Amazon Relational Database Service (RDS)

        * Allows to set up a conventional relational database (Postgres/MySQL) on Amazon computers to port existing applications
    - Amazon Aurora Service
        * Compatible with MySQL
        * Higher availability, performance, and resilience than RDS
        * Can scale to many TBs, replicate across data centers
        * Can create many read replicas to support large number of concurrent reads
    - Google's Cloud SQL
        * Has a Spanner system that is globally distributed providing ACID transactions and SQL semantics with high scaling and availability
    - Azure's SQL database
- Warehouse analytics
    - Amazon Redshift
        * Data warehouse system
        * Supports high performance execution of analytic and reporting workloads against large datasets
    - Google BigQuery
        * Petascale data warehouse
        * Fully distributed and replicated
        * Supports SQL query semantics
    - Azure Data Lake
        * Full suite of data analytics tools
        * Built on open source YARN and WebHDFS platforms
- Graphs and more
    - Messaging services
        * Allow applications to send/receive messages using *publish/subscribe* semantics
        * One application waits on a queue for a message to arrive
        * Other applications prepare the message and send it to queue
    - Amazon's Titan
        * Extension to DynamoDB
        * Supports graph databases
    - Google
        * Supports open source database Cayley
    - Azure Graph Engine
        * Distributed, in-memory, large graph processing system
- OpenStack storage services and Jetstream
    - Supports only a few standard storage services: object storage, block storage, and file system storage
    - OpenStack object storage service Swift
        * Implements a REST API
        * Allows users to store, delete, manage permissions of, and associate metadata with immutable unstructured data objects located within containers
        * Objects replicated across multiple storage servers for fault tolerance and performance reasons; may be accessed from anywhere

- OpenStack Shared File Systems service
  - Implements a file system model in cloud environment
  - Users interact with the service by mounting remote file systems, called *shares* on their virtual machine instances
  - Users can creates shares, configure the file system protocol system supported, manage access to shares, delete shares, and configure rate limits and quotas
  - Shares may be mounted on any number of client machines using NFS, CIFS, GlusterFS, or HDFS drivers
  - Shares may be accessed only from VM instances running on the OpenStack cloud
- Jetstream
  - Operated as a part of the XSEDE supercomputer project `xsede.org`
  - Runs OpenStack object store, based on Ceph, implementing the Swift API
  - Primary user interaction through a system called Atmosphere
  - Atmosphere
    - Designed to manage VMs, data, and visualization tools
    - Provides a volume management system to mount external volumes on VMs
  - Operates Globus identity, group, and file management services

Using Cloud Storage Services

Two access methods: Portals and APIs

- Possible to accomplish most tasks by using a few mouse clicks
  - Not good for repetitive tasks; managing hundreds of data objects
  - REST APIs allow to access storage services programmatically
  - Access APIs via SDKs
- REST APIs and SDKs from different providers are not identical
  - Standardization efforts under progress
  - CloudBridge and Apache Libcloud `Libcloud.apache.org` as Python SDK
  - May not be able to cover all capabilities of individual platforms
- Build data sample collection in cloud
  - Collection of data samples on PC to be accessed by collaborators
  - Four items of metadata for each sample: item number, creation date, experiment ID, text string comment
  - Upload to cloud storage and create a searchable table, also hosted in the cloud, containing metadata and cloud storage URL for each object (Figure 3.1)
  - Each data sample in binary format on PC
  - Associated metadata in CSV file, with one line per item, also on PC
  - CSV file format for each line

item ID, experiment ID, date, filename, comment string

Using Amazon cloud storage services

- Use S3 to store blobs and DynamoDB to store the table
- Need Amazon key pair (access key plus secret key) obtained from Amazon IAM Management Console

- Create a new user; select *create access key* button to create security credentials and download it (Figure 3.2)
- Create the required S3 bucket and upload blobs to that bucket from the Amazon web portal
- Use Amazon Python Boto3 SDK to upload multiple blobs
  - Boto3 considers each service to be a *resource*
  - Create an S3 resource object to use the S3 system
  - Specify credentials obtained from the IAM Management Console
    * Provide credentials to Python program
    * As named parameters to resource instance creation function

      ```
      import boto3
      s3 = boto3.resource('s3',
          aws_access_key_id='Your Access Key',
          aws_secret_access_key='Your Secret Key' )
      ```
    * Problem with the approach
      · Security credentials in clear text format
      · May be fixed creating a directory `$HOME/.aws` that contains two protected files
        1. `config` – contains your default Amazon region
        2. `credentials` – contains your access and secret keys
      · With those two files, access and secret keys are not needed
  - After creating S3 resource object, create the S3 bucket `datacont` to store data objects

    ```
    import boto3
    s3 = boto.resource ( 's3' )
    s3.create_bucket(Bucket = 'datacont',
        CreateBucketConfiguration = {'LocationConstraint': 'us-west-2'})
    ```
    * Second argument to `create_bucket` is optional (and is also the default if no region is specified)
    * At present, there are 17 regions operated by Amazon listed at
      https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html
  - Load data objects into the new bucket

    ```
    # Upload a file, 'test.jpg' into the newly created bucket
    s3.Object('datacont', 'test.jpg').put(
        Body=open('/home/mydata/test.jpg', 'rb'))
    ```
  - Create the DynamoDB table to store metadata and references to S3 objects
    * Create the table by defining a special key composed of a `PartitionKey` and a `RowKey`
    * NoSQL systems such as DynamoDB are distributed over multiple storage devices to enable the construction of extremely large tables, accessible in parallel by many servers
    * Table's aggregate bandwidth is multiple of number of storage devices
    * DynamoDB distributes data by rows
      · Every element in the row is mapped to the same device
      · Device determined by `PartitionKey` which is hashed to an index that determines the physical storage device in which the row resides
      · `RowKey` specifies that items are stored in order sorted by the `RowKey` value
    * Use the following code to create DynamoDB table

      ```
      dyndb = boto3.resource ( 'dynamodb', region_name='us-west-2' )

      # First time definition of table
      table = dyndb.create_table(
          TableName='DataTable',
      ```

```
        KeySchema=[
            { 'AttributeName': 'PartitionKey', 'KeyType', 'HASH' },
            { 'AttributeName': 'RowKey',       'KeyType', 'RANGE' }
        ],
        AttributeDefinitions=[
            { 'AttributeName': 'PartitionKey', 'AttributeType': 'S' },
            { 'AttributeName': 'RowKey',       'AttributeType': 'S' }
        ],
    )

    # Wait for the table to be created
    table.meta.client.get_waiter('table_exists').wait(TableName='DataTable')

    # If the table has been previously defined
    # table = dyndb.Table("DataTable")
```

  - Read data from CSV file
    * CSV file format

      ```
      itemID, experimentID, date, filename, comment
      ```

    * URL for the data file should be publicly readable – indicated via `ACL='public-read'`

      ```
      import csv
      urlbase = "https://s3-us-west-2.amazonaws.com/datacont/"
      with open('\path-to-your-data\experiments.csv','rb') as csvfile:
          csvf = csv.reader(csvfile.delimiter=',',quotechar'|')
          for item in csvf:
              body = open('path-to-your-data\datafiles\\'+item[3]).put(Body=body)
              md = s3.Object('datacont', item[3]).Acl()
                  .put(ACL='public-read')
              url=urlbase + item[3]
              metadata_item={'PartitionKey': item[0], 'RowKey': item[1],
                  'description': item[4], 'date':item[2], 'url':url}
              table.put_item(Item=metadata_item)
      ```

Using Microsoft Azure storage services

- Amazon account ID is defined by a pair – your access key and your secret key
- Azure account defined by your personal ID and a subscription ID
  - Personal ID may be your email address – public
  - Subscription ID should be kept secret
- Implement example using Azure standard blob storage and Table service
  - Each row has fields `PartitionKey`, `RowKey`, `comments`, `date`, and `URL` just as in Amazon DynamoDB
  - `RowKey` is a unique integer for each row
    * Unique global identifier for the row
  - `PartitionKey` used as a hash to locate a row in specific storage device
- Storage services
  - In Amazon S3, you create buckets and then, create blobs within a bucket