Cloud Storage

Big Data and Cloud Computing (CC4093)

Eduardo R. B. Marques, DCC/FCUP

Cloud storage

We provide an overview of the main storage models used with cloud computing, and example cloud service offerings including:

- File systems
- Object stores (buckets)
- Databases: relational (SQL-based) and other kinds (“NoSQL”)
- Data warehousing

Reference → most of the topics in these slides are also covered in chapters 2 and 3 of Cloud Computing for Science and Engineering.

Local file systems

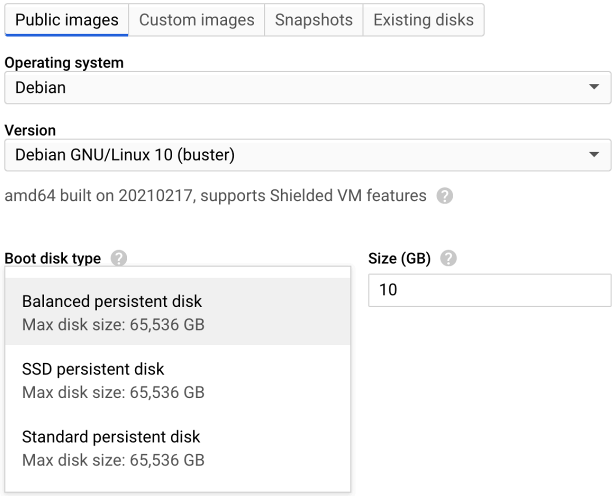
Characteristics:

- hierarchical organisation of files within directories
- uses an entire disk or a disk partition with fixed capacity
- data is read/written by a single host
- programmatic access by standard APIs (e.g., the POSIX interface)

Advantage → most programs are designed to work with files, and use the file system hierarchy (directories) as a natural means to organize information - files/directories have no associated "data schema" though.

Disavantadges → Shared access by multiple hosts is not possible, and the volume of data is bound by the file system capacity. Distributed file systems, discussed next, can handle shared access and mitigate volume restrictions.

Local file systems (cont.)



Operating system
Debian

Version
Debian GNU/Linux 10 (buster)

amd64 built on 20210217, supports Shielded VM features

Boot disk type

Balanced persistent disk
Max disk size: 65,536 GB

SSD persistent disk
Max disk size: 65,536 GB

Standard persistent disk
Max disk size: 65,536 GB

Size (GB)
10

Example: configuration of a VM book disk in Compute Engine

Network attached storage

Characteristics:

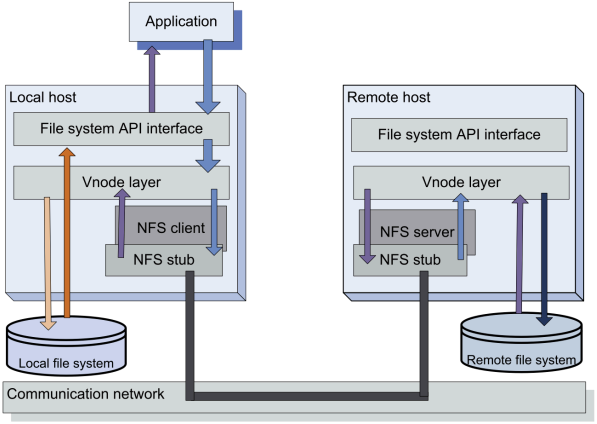
- Files are physically stored in a server host. The file system can be attached over the network by other hosts, and accessed transparently (i.e. the same way as local file systems).
- Data consistency is managed by a network protocol such as NFS (Networked File System) and SMB (Server Message Block).

Advantages → data sharing by multiple hosts, and typically higher storage capacity (installed at server hosts)

Example cloud services → NFS: Amazon EFS, Google File Store, Azure Files (Azure Files also supports SMB).

Beyond shared storage, a note on parallel file systems → file systems such as Lustre or HDFS are designed for parallel computing using computer clusters. We will talk about HDFS later in the semester.

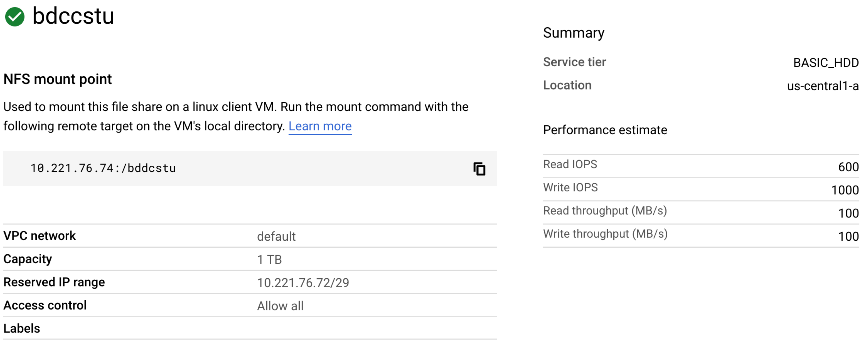
Distributed File Systems (cont.)



NFS client-server interaction

Image source: “Cloud Computing: Theory and Practice 2nd ed.” by D. C. Marinescu

Distributed File Systems (cont.)



✓ bdccstu

NFS mount point

Used to mount this file share on a linux client VM. Run the mount command with the following remote target on the VM's local directory. Learn more

```
10.221.76.74:/bddcstu
```

VPC network	default
Capacity	1 TB
Reserved IP range	10.221.76.72/29
Access control	Allow all
Labels	

Summary

Service tier	BASIC_HDD
Location	us-central1-a

Performance estimate

Read IOPS	600
Write IOPS	1000
Read throughput (MB/s)	100
Write throughput (MB/s)	100

Example: NFS configuration using Google File Store

Buckets

An **object store**, also known as **bucket** (in AWS and GCP), is a container for unstructured binary objects. It can be accessed over the Internet without any logical association to specific storage facility. An object in a bucket:

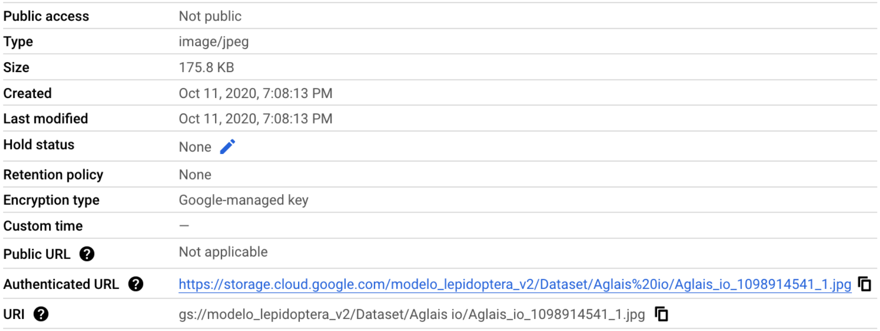
- is identified by an URI (uniform resource identifier)
  e.g. `gs://bucket_name/object_name` in Google Cloud Storage or `s3://bucket_name/object_name` in Amazon S3
- also has an associated HTTP URL derived from the URI
- is not tied to any hierarchical file system organisation even though URI may contain “paths”
  e.g. `gs://bucket_name/path/to/object`.
- cannot be modified once created, only deleted or updated (whole-object “get”, “put”, “list”, and “delete” operations);
- can have a very large size (e.g. 5 TB in GCS and S3) and the container bucket has no site limit;
- has associated meta-data regarding diverse aspects (e.g., storage class, access permissions).

File systems vs. buckets

- File systems
  - tied to a particular hierarchical organisation and storage device
  - files can be modified or accessed arbitrarily
  - size limit determined by underlying storage facility
  - accessed by programs running in a host that mounts the file system
  - can be mounted efficiently over the network by a host
- Buckets
  - data storage not bound logically to any storage facility
  - whole-object operations
  - has no size limit even if contained objects do
  - accessible over the Internet
  - can be mounted over the network by a host (e.g. using gcsfuse) but with performance limitations

The logical simplicity of the bucket model allows easy replication with no synchronization constraints, and flexible/scalable use in diverse use cases (cloud apps, content delivery, backup, etc). Example: public data sets of Landsat/Sentinel satellite imagery are available, and used by Google Earth Engine.

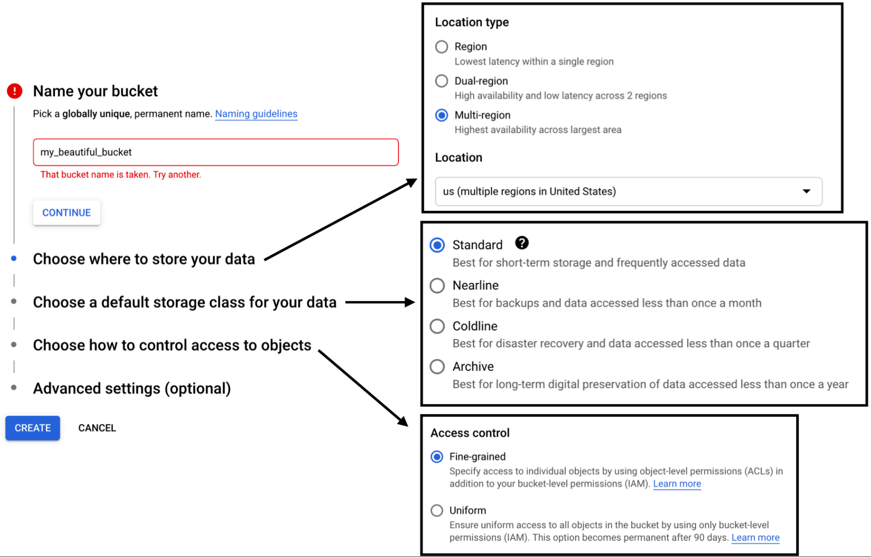
Buckets in GCP



Public access	Not public
Type	image/jpeg
Size	175.8 KB
Created	Oct 11, 2020, 7:08:13 PM
Last modified	Oct 11, 2020, 7:08:13 PM
Hold status	None
Retention policy	None
Encryption type	Google-managed key
Custom time	—
Public URL	Not applicable
Authenticated URL	https://storage.cloud.google.com/modelo_lepidoptera_v2/Dataset/Aglais%20io/Aglais_io_1098914541_1.jpg
URI	gs://modelo_lepidoptera_v2/Dataset/Aglais io/Aglais_io_1098914541_1.jpg

Example from Google Cloud Platform

Bucket creation in GCP



Name your bucket

Pick a **globally unique**, permanent name. Naming guidelines

my_beautiful_bucket

That bucket name is taken. Try another.

CONTINUE

- Choose where to store your data
- Choose a default storage class for your data
- Choose how to control access to objects
- Advanced settings (optional)

CREATE CANCEL

Location type

Region
Lowest latency within a single region

Dual-region
High availability and low latency across 2 regions

[✓ Multi-region]
Highest availability across largest area

Location

us (multiple regions in United States)

[✓ Standard]
Best for short-term storage and frequently accessed data

Nearline
Best for backups and data accessed less than once a month

Coldline
Best for disaster recovery and data accessed less than once a quarter

Archive
Best for long-term digital preservation of data accessed less than once a year

Access control

[✓ Fine-grained]
Specify access to individual objects by using object-level permissions (ACLs) in addition to your bucket-level permissions (IAM). Learn more

Uniform
Ensure uniform access to all objects in the bucket by using only bucket-level permissions (IAM). This option becomes permanent after 90 days. Learn more

Basic parameters → a globally unique **name**, **region placement**, and **storage class**.

Bucket creation in GCP and billing

Parameters:

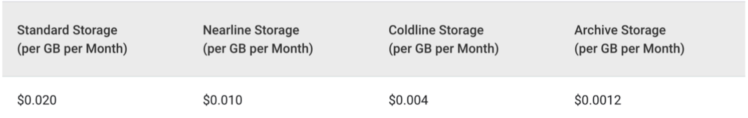
- Name must be globally unique in the scope of GCP.
- Region placement is a choice between low latency in a single region, two regions, or multi-region in the same continent.
- Storage class:
  - Standard: frequently accessed data
  - Near-line, Cold-line, Archive: infrequently accessed data

Billing:

- Buckets are billed based on storage usage, network usage, and number of bucket operations
- Standard storage class: more expensive in terms of storage usage, cheaper regarding bucket operations.
- Region settings also affects billing.

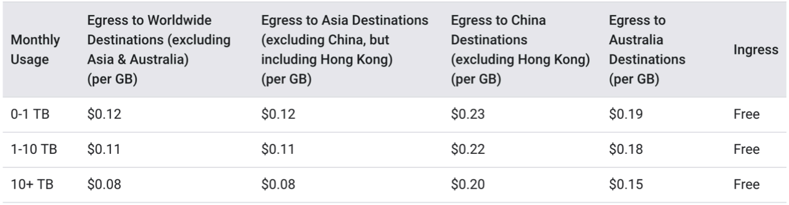
Bucket creation in GCP and billing (cont.)

storage usage



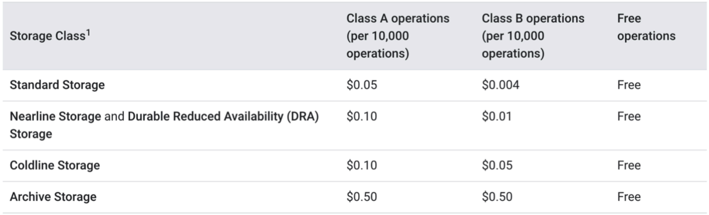
Standard Storage (per GB per Month)	Nearline Storage (per GB per Month)	Coldline Storage (per GB per Month)	Archive Storage (per GB per Month)
$0.020	$0.010	$0.004	$0.0012

network usage



Monthly Usage	Egress to Worldwide Destinations (excluding Asia & Australia) (per GB)	Egress to Asia Destinations (excluding China, but including Hong Kong) (per GB)	Egress to China Destinations (excluding Hong Kong) (per GB)	Egress to Australia Destinations (per GB)	Ingress
0-1 TB	$0.12	$0.12	$0.23	$0.19	Free
1-10 TB	$0.11	$0.11	$0.22	$0.18	Free
10+ TB	$0.08	$0.08	$0.20	$0.15	Free

operations



Storage Class[1]	Class A operations (per 10,000 operations)	Class B operations (per 10,000 operations)	Free operations
Standard Storage	$0.05	$0.004	Free
Nearline Storage and **Durable Reduced Availability (DRA) Storage**	$0.10	$0.01	Free
Coldline Storage	$0.10	$0.05	Free
Archive Storage	$0.50	$0.50	Free

From Cloud Storage pricing, Feb. 2021

Access to GCP buckets

- Google Cloud console
- Bucket objects can be downloaded using HTTPS (bucket objects have URLs)
- gsutil (Google Cloud SDK utility)
- REST APIs in JSON and XML.
- Client libraries in several programming languages (also bundled with the Google Cloud SDK)

Access to GCP buckets using Python

```python
# Imports the Google Cloud client library
from google.cloud import storage

# Instantiates a client
storage_client = storage.Client()

# The name for the new bucket
bucket_name = "my-new-bucket"

# Creates the new bucket
bucket = storage_client.create_bucket(bucket_name)

print("Bucket {} created.".format(bucket.name))
```

Reference: google-cloud-storage library

Access to GCP buckets using Python (cont.)

```python
def download_blob(bucket_name,
                  source_blob_name,
                  destination_file_name):
    """Downloads a blob from the bucket."""
    # bucket_name = "your-bucket-name"
    # source_blob_name = "storage-object-name"
    # destination_file_name = "local/path/to/file"
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(source_blob_name)
    blob.download_to_filename(destination_file_name)
```

Reference: google-cloud-storage library

Databases

Database $\rightarrow$ structured collection of data concerning entities in the real-world and their relationship. The structure of data is many times described by a **data schema** (also called a data model).

Database management system (DBMS) $\rightarrow$ software system responsible by managing data schemas and the associated data, application interface using a **query language**, supporting concurrent **transactions**, and handling **crash recovery**.

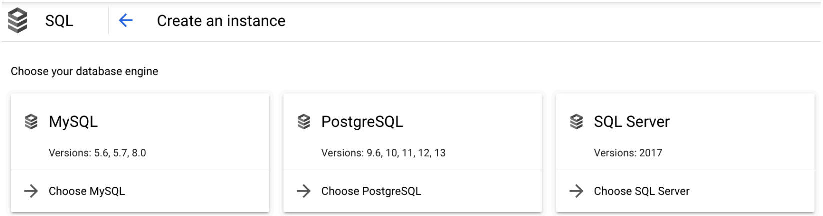
Types of databases:

- **Relational DBMS** (SQL-based) $\rightarrow$ data organised as tables that can be manipulated using SQL (Structured Query Language).
- **"NoSQL" databases** $\rightarrow$ "non-SQL" or "not only SQL" databases may use other models for data organisation (key-value, document store, graph-based, ...) and/or query languages.

Relational databases - use of SQL

```sql
-- Table creation
CREATE TABLE CUSTOMER(
  Id PRIMARY KEY AUTO_INCREMENT,
  Login UNIQUE VARCHAR(16) NOT NULL,
  Name VARCHAR(64) NOT NULL,
  Phone INT NOT NULL
  Country VARCHAR(64) NOT NULL );
-- Data changes
INSERT INTO CUSTOMER(Login,Name,Phone,Country)
VALUES('bdcc','Alonsus Big' 9299334, 'Portugal');
UPDATE CUSTOMER SET Phone = 93949444 WHERE Login='bdcc';
DELETE FROM CUSTOMER WHERE Country = 'Spain';
-- Queries
SELECT Login,Country FROM CUSTOMER WHERE Id='1234';
SELECT * FROM CUSTOMER
WHERE Country = 'China' AND Phone IS NOT NULL
ORDER BY Login;
```

Relational databases - DBaaS offerings



Choose your database engine

MySQL

Versions: 5.6, 5.7, 8.0

Choose MySQL

PostgreSQL

Versions: 9.6, 10, 11, 12, 13

Choose PostgreSQL

SQL Server

Versions: 2017

Choose SQL Server

General-purpose and popular DBMSs ´like MySQL, PostgreSQL, or Microsoft SQL Server are offered through DBaaS (Database as a Service) solutions like e.g. Google Cloud SQL, Amazon RDS, or Azurel SQL.

Other offerings are oriented towards scalable operation in cloud computing, e.g., Google Spanner or Amazon Aurora.

Databases and ACID transactions

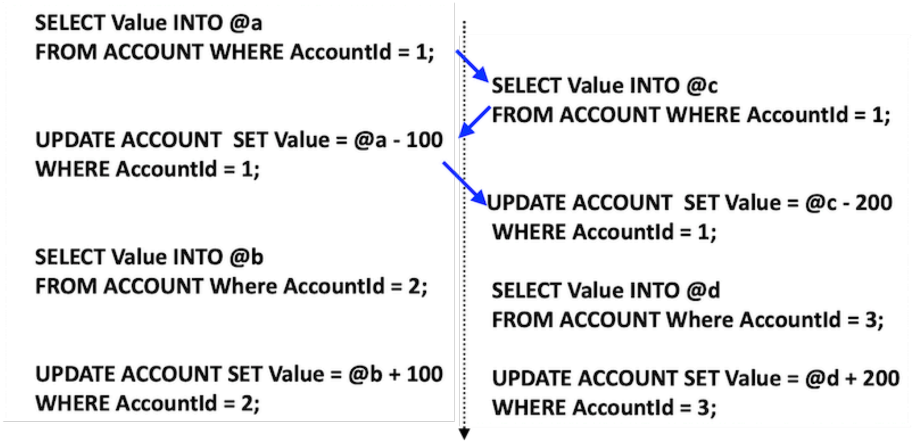
Transaction = sequence of database operations

- **Atomicity**: the entire transaction succeeds or fails.
- **Consistency**: a transaction preserves database consistency.
- **Isolation**: concurrent transactions do not interfere with each other.
- **Durability**: if a transaction succeeds, its changes persist in the database.

ACID guarantees are often conflicting, especially if a database must scale to accommodate large volumes of data or concurrent accesses, and/or provide high-performance access to data.

For example, the SQL standard defines several “isolation levels” related to relaxations of the isolation requirement, which in turn may compromise consistency and atomicity.

Databases and ACID transactions (cont.)

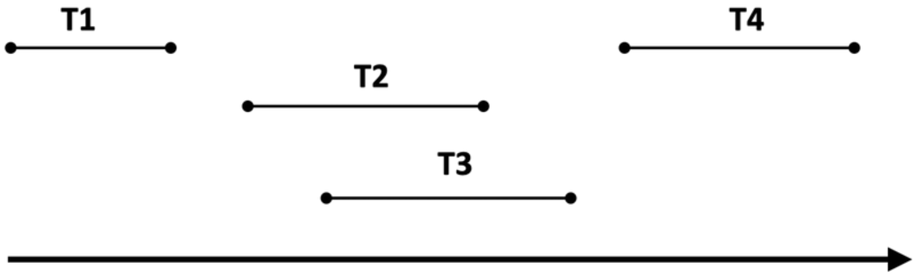


```
SELECT Value INTO @a
FROM ACCOUNT WHERE AccountId = 1;

UPDATE ACCOUNT  SET Value = @a - 100
WHERE AccountId = 1;

SELECT Value INTO @b
FROM ACCOUNT Where AccountId = 2;

UPDATE ACCOUNT SET Value = @b + 100
WHERE AccountId = 2;
```

```
SELECT Value INTO @c
FROM ACCOUNT WHERE AccountId = 1;

UPDATE ACCOUNT  SET Value = @c - 200
WHERE AccountId = 1;

SELECT Value INTO @d
FROM ACCOUNT Where AccountId = 3;

UPDATE ACCOUNT SET Value = @d + 200
WHERE AccountId = 3;
```

Two transactions involving money transfer between bank accounts: **1 → 2 with a value of 100** (left) and **1 → 3 with a value of 200** (right). If they are not properly isolated we may have that for example that the balance of account 1 is decremented by 200 rather than 300.

Databases and ACID transactions (cont.)



Concurrent transactions must appear as if they executed in sequence i.e., the DBMS must ensure they are **serializable**.

In the example above with 4 transactions, where T2 and T3 are concurrent, the logical effect should be as if we executed the transactions in one the following orders: $\mathbf{T1} \rightarrow \mathbf{T2} \rightarrow \mathbf{T3} \rightarrow \mathbf{T4}$ or $\mathbf{T1} \rightarrow \mathbf{T3} \rightarrow \mathbf{T2} \rightarrow \mathbf{T4}$

Distributed databases

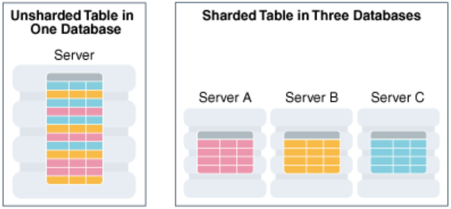

Unsharded Table in One Database
Server
Sharded Table in Three Databases
Server A
Server B
Server C

Image source: Overview of Oracle sharding, Oracle corp.

To cope with scalable operation for large amounts of data and accesses, distributed databases employ techniques such as sharding (data is partitioned across different hosts, illustrated above) or replication (data is mirrored across different hosts).

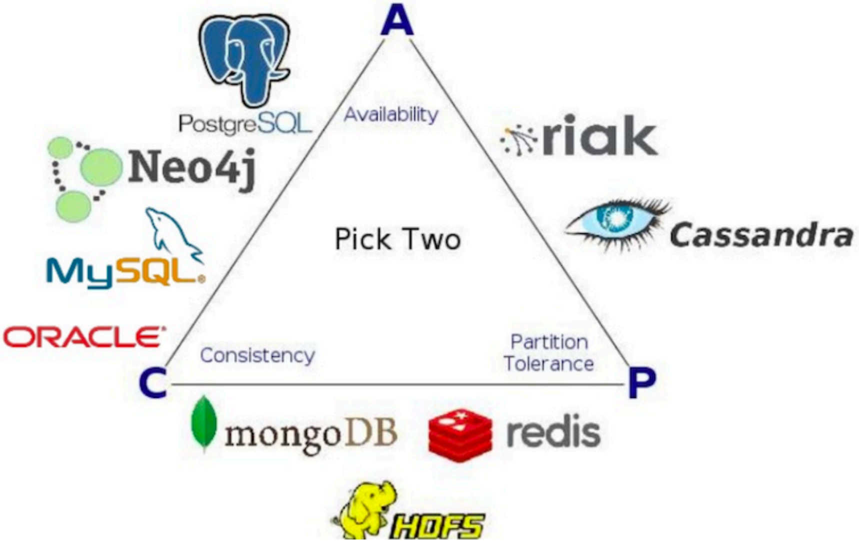
Distributed operation presents new challenges, though ...

Distributed databases and the CAP theorem

CAP Theorem (also known as Brewer's theorem) → an influential theoretical result for distributed databases that states that **a system cannot guarantee all of the three** following properties:

- **Consistency**: all computers see the same data at the same time.
- **Availability**: every request always gets a response.
- **Partition tolerance**: the system continues to operate in the presence of network failures.

Distributed databases and the CAP theorem (cont.)



CAP? Choose two! Database systems fall under one the CA, CP, or AP categories.

NoSQL databases

- NoSQL is a general designation for database systems that are not relational.
- Data may be organised according to several paradigms: key-value store, document store, or graph-based. Some systems also support a subset of SQL or SQL-like query languages.
- Data may have no schema, or be “self-describing”.
- A key motivation was the simplicity of implementation in a distributed setting. Strict consistency is often sacrificed at the expense of availability and partition tolerance.

NoSQL database systems

Key-value stores defines two abstract operations: `put(k,v)`: stores value `v` in association to key `k`, and; `get(k)`: get value associated to key `k`. The types of keys and values are often arbitrary, i.e., treated as byte sequences by the storage system.

Document stores are specialised key-value stores that operate on data, called documents, that have a "self-describing" hierarchical structure (e.g., as in XML, JSON). DB operations can be formulated in terms of the "self-describing" schema, sometimes using SQL-like languages. Examples → MongoDB, Google FireStore.

Graph databases organizes data in terms of graphs, such that nodes represent entities and edges represent relationships between entities. Example → Neo4j.

Multi-paradigm NoSQL databases → ArangoDB and Azure Cosmos are examples that support key, document, or graph-based operations.

We only cover some examples of key-value stores next.

BigTable

BigTable from Google is a key-value cloud storage system for operating large volumes of data. It underlies several Google services like Google Earth, Google Analytics and in fact other NoSQL databases like Google FireStore.

- High availability, scalable, fault-tolerant (AP class). A single BigTable database can scale up to thousands of servers. Data is split / replicated on a cluster of machines.
- Consistency: does not support transactions, only atomic reads and writes for individual data items.

BigTable (cont.)

BigTable is a **column wide-store**, a special type of key-value store.

As in relational databases, BigTable supports the concepts of table, row, and column, but with significant differences. Table = { rows }, however:

- Rows have a unique key.
- Rows in the same table can have distinct columns.
- Columns are grouped into different **families**, such that different column families in a database are stored separately.
- A data item is indexed by both the row key and the column family.
- Data is queried using frameworks like MapReduce (we'll talk about this later).

BigTable (cont.)

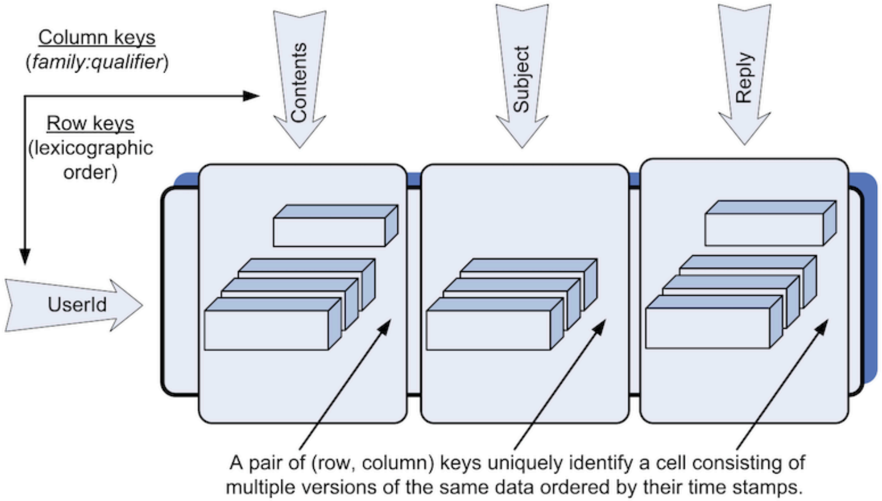

Column keys
(family:qualifier)
Row keys
(lexicographic
order)
Contents
Subject
Reply
UserId
A pair of (row, column) keys uniquely identify a cell consisting of
multiple versions of the same data ordered by their time stamps.

Image source: “Cloud Computing: Theory and Practice 2nd ed.” by D. C. Marinescu

BigTable (cont.)

```
from gcloud import bigtable
clientbt = bigtable.Client(admin=True)
clientbt.start()
instance = clientbt.instance('cloud-book-instance')
table = instance.table('book-table2')
table.create()
column_family = table.column_family('cf')
column_family.create()
row= table.row('key_value')
row.set_cell('cf', 'experiment', 'exp1')
row.set_cell('cf', 'date', '6/6/16')
row.set_cell('cf', 'link', 'http://some_location')
row.commit()
row_data = table.read_row('key_value')
cells = row_data.cells['cf']
```

Reference: Python Client for Google Cloud Bigtable

In-memory key-value stores

```
from pymemcache.client.base import Client
mc = Client('my-memcached-server.com')
mc.set('foo', 'bar')
result = mc.get('foo')
```

```
import redis
r = redis.Redis(host='localhost', port=6379, db=0)
r.set('foo', 'bar')
data = r.get('foo')
```

Memcached and Redis are two popular distributed in-memory key-value stores. (Redis also supports persistence and a number of other features beyond the raw key-value store functionality).

A typical use case → act as a cache for frequently accessed data in another database system. Both can be deployed using the Google Cloud Memorystore service.

In-memory key-value stores (cont.)

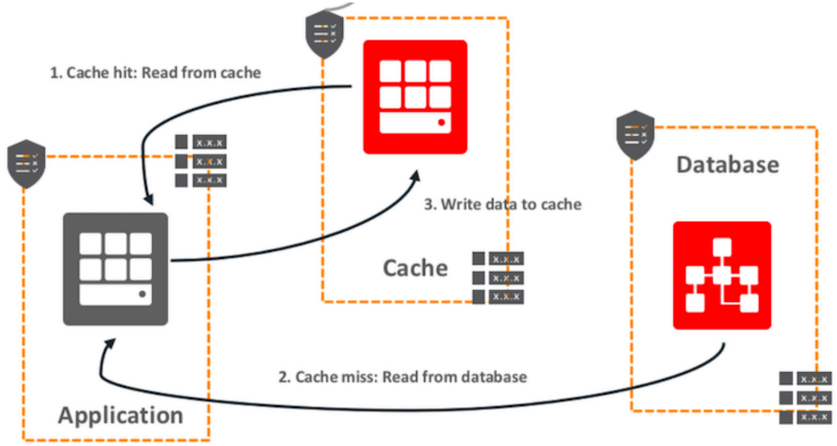

1. Cache hit: Read from cache
3. Write data to cache
2. Cache miss: Read from database
Application
Cache
Database
x.x.x

Image source: Deploying a HighlyAvailable Distributed Caching Layer on Oracle Cloud Infrastructure using Memcached & Redis, Oracle white paper, 2018.

What database should I use?

It depends on your use case. The following fragment from the Google BigTable documentation offers some advice:

- “Cloud Bigtable is not a relational database. It does not support SQL queries, joins, or multi-row transactions.”
- “If you need full SQL support for an online transaction processing (OLTP) system, consider Cloud Spanner or Cloud SQL.”
- “If you need interactive querying in an online analytical processing (OLAP) system, consider BigQuery.” - we’ll cover BigQuery next.
- “If you need to store highly structured objects in a document database, with support for ACID transactions and SQL-like queries, consider Firestore.”
- “For in-memory data storage with low latency, consider Memorystore.”

Data warehouses

Data warehouses → systems used for storing and querying large datasets. They are used by **data analytics** tools whose purpose is to run intensive queries over the data in scalable manner.

In contrast to DBMS:

- stored data is immutable (write-once or append-only)
- system must scale for extremely large datasets (up to penta/hexa-byte scale)
- queries typically involve collection of records (aggregate queries)

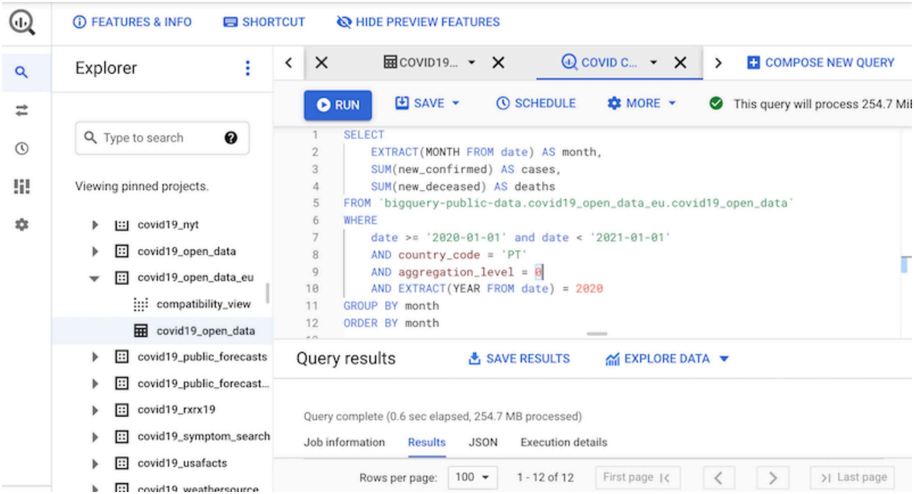
Example warehouse services → Google BigQuery, Amazon Redshift, and Azure Data Lake.

Google BigQuery

Essential characteristics:

- BigQuery is serverless. There is no need to configure or maintain servers to use BigQuery.
- Users just need to focus on data set creation and queries using assorted APIs (e.g., REST-based or client APIs for many programming languages).
- Queries are specified using standard SQL.
- Billing is defined by the number of queries and volumes of data involved in queries.

BigQuery UI



```
SELECT
    EXTRACT(MONTH FROM date) AS month,
    SUM(new_confirmed) AS cases,
    SUM(new_deceased) AS deaths
FROM `bigquery-public-data.covid19_open_data_eu.covid19_open_data`
WHERE
    date >= '2020-01-01' and date < '2021-01-01'
    AND country_code = 'PT'
    AND aggregation_level = 0
    AND EXTRACT(YEAR FROM date) = 2020
GROUP BY month
ORDER BY month
```

Query complete (0.6 sec elapsed, 254.7 MB processed)

Query example over a BigQuery public dataset

Access to Google BigQuery using Python

Query example

```
import google.cloud.bigquery as bq
client = bq.Client(project=projectId)
query = client.query('''
   SELECT count,
          origin, org.name AS origin_name,
          destination, dst.name AS destination_name
   FROM   `bdcc.vegas.flights`
   JOIN   `bdcc.vegas.airports` org ON(origin=org.iata)
   JOIN   `bdcc.vegas.airports` dst ON(destination=dst.iata)
   WHERE   org.state = 'NY'
   ORDER BY count DESC, origin, destination''')
pandas_df = query.to_dataframe()
```

Query example

Reference: Python Client for Google BigQuery

Access to Google BigQuery using Python (cont.)

```
import google.cloud.bigquery as bq
client = bq.Client(project=projectId)
table = bq.Table(table_name)
table.schema = (
   bq.SchemaField("movieId", "INTEGER", "REQUIRED"),
   bq.SchemaField("title",  "STRING", "REQUIRED"),
   bq.SchemaField("year", "INTEGER", "REQUIRED"),
   bq.SchemaField("imdbId", "INTEGER", "REQUIRED")
)
pandas_df = ...
load_job = client.load_table_from_dataframe(pandas_df, table)
while load_job.running():
   time.sleep(0.5) # wait ...
```

Table creation example

Reference: Python Client for Google BigQuery

Google BigQuery - other features

- BigQuery ML enables users to create and execute machine learning models in BigQuery by using standard SQL queries
- BigQuery GIS native support for geospatial analysis that is required by Geographic Information Systems (GIS).
- BigQuery BI Engine is a fast, in-memory analysis service for integration with other tools (e.g., visual data analytics in Google Data Studio, Looker, Tableau).