Big Data, Techniques and Platforms
Class 5/8: Data Distribution and CAP

Francesca Bugiotti

CentraleSupélec

October 17, 2023

Objectives

- Class 4: Data distribution models
- Class 4: Consistency in distributed databases
- Availability in distributed databases
- CAP theorem

Plan

Summary

Data Science

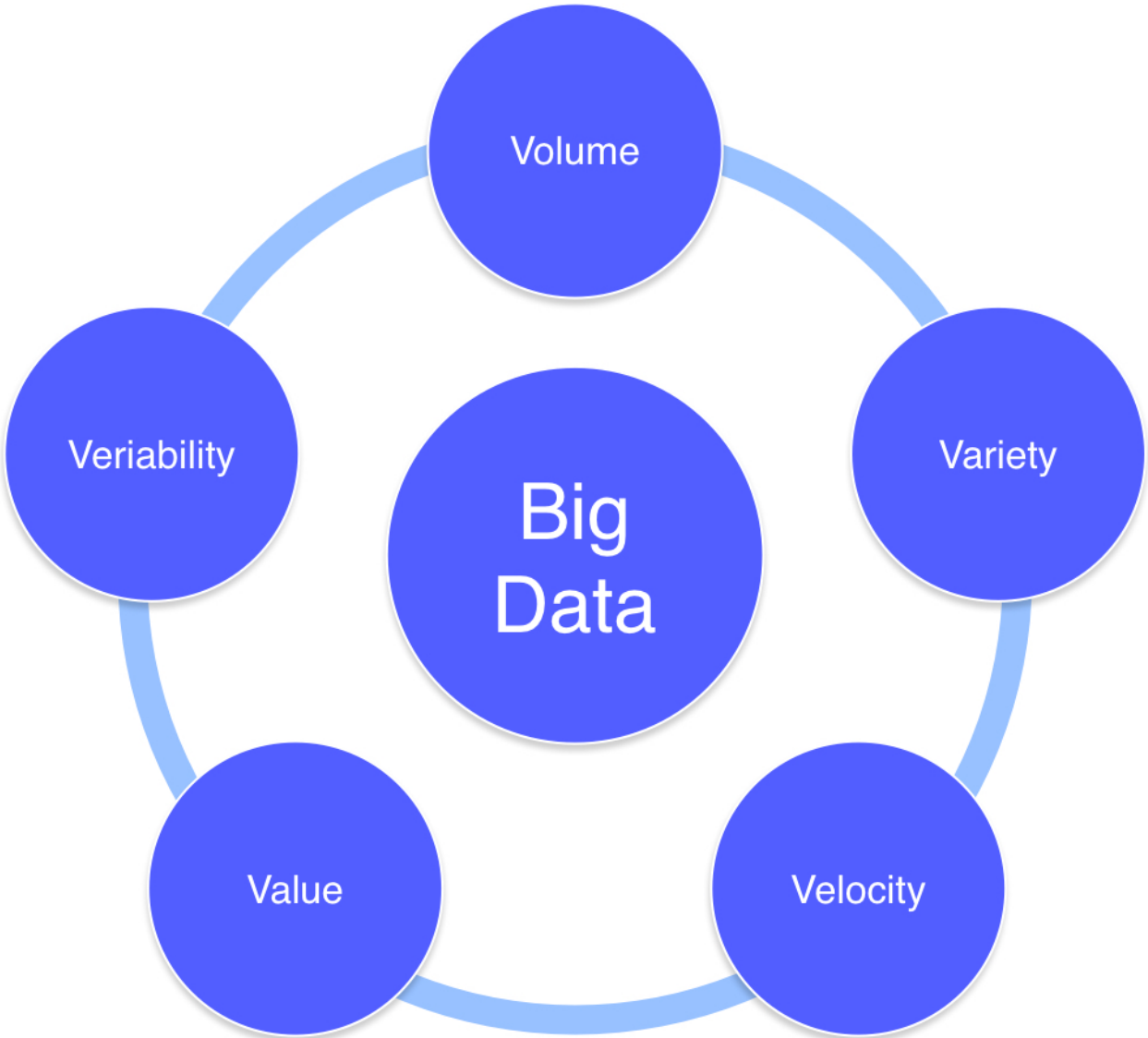
- Data
  - Common devices and specialized devices
  - New data sources producing torrent of data
- Cloud Computing
  - Computing anywhere and anytime
  - On-demand Computing

Big data is not just "Big"

Combining data from different sources

- Machines
  - real-time data, sensor data, trackers, streaming, etc.
- People
  - social media, personal documents, etc.
- Organizations
  - structured knowledge bases, data-warehouses, etc.

Big Data: the 5 Vs



MODERN DATA SCIENTIST

Data Scientist, the sexiest job of the 21th century, requires a mixture of multidisciplinary skills ranging from an intersection of mathematics, statistics, computer science, communication and business. Finding a data scientist is hard. Finding people who understand who a data scientist is, is equally hard. So here is a little cheat sheet on who the modern data scientist really is.

MATH & STATISTICS

- Machine learning
- Statistical modeling
- Experiment design
- Bayesian inference
- Supervised learning: decision trees, random forests, logistic regression
- Unsupervised learning: clustering, dimensionality reduction
- Optimization: gradient descent and variants

PROGRAMMING & DATABASE

- Computer science fundamentals
- Scripting language e.g. Python
- Statistical computing packages, e.g., R
- Databases: SQL and NoSQL
- Relational algebra
- Parallel databases and parallel query processing
- MapReduce concepts
- Hadoop and Hive/Pig
- Custom reducers
- Experience with xaaS like AWS

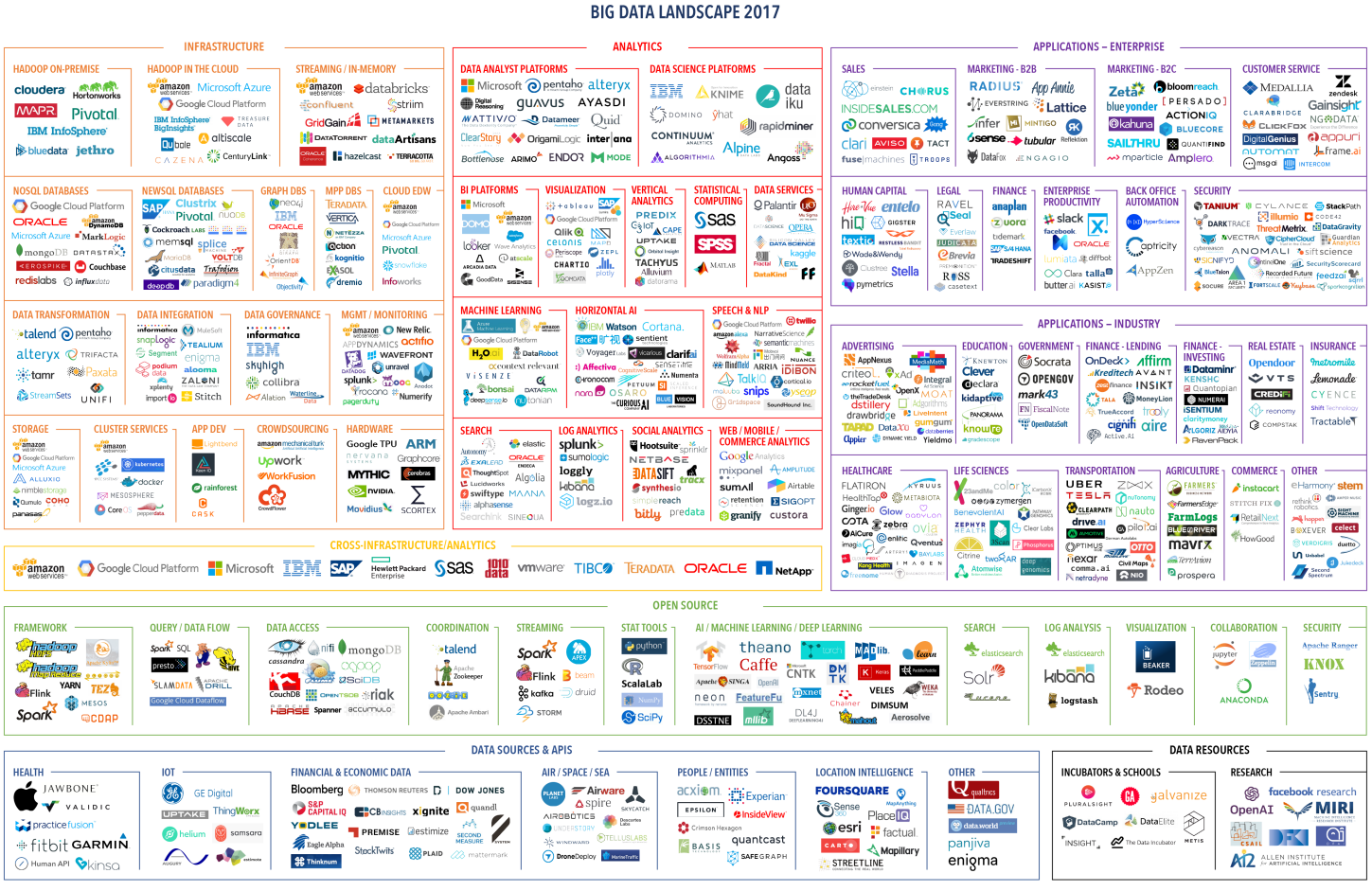
DOMAIN KNOWLEDGE & SOFT SKILLS

- Passionate about the business
- Curious about data
- Influence without authority
- Hacker mindset
- Problem solver
- Strategic, proactive, creative, innovative and collaborative

COMMUNICATION & VISUALIZATION

- Able to engage with senior management
- Story telling skills
- Translate data-driven insights into decisions and actions
- Visual art design
- R packages like ggplot or lattice
- Knowledge of any of visualization tools e.g. Flare, D3.js, Tableau

Technologies



V2 – Last updated 5/3/2017

 mattturck.com/bigdata2017

FIRSTMARK EARLY STAGE VENTURE CAPITAL

Bakery

Baker - Chef

Baker - Chef

Ingredients

Stockage Utils

Utils

Recipes


CIAMBELLONE SALATO
EMPANADA AL FORNO
(con ripieno di carne)
PREPARAZIONE
PREPARAZIONE


Galettes à la noix de coco
La pâte :
- 250 gr de beurre ramolli
- 1 verre de sucre glace
- 3 œufs
- 1 paquet de levure chimique
- 1 C. à café de vanille
- 1 verre de maïzena
- Farine (SIM)
La décoration :
- Fruits confits
- Noix de coco colorée
- Confiture
1- Préparer les ingrédients nécessaires.
2- Dans un récipient, mélanger le beurre, le sucre, la vanille, la maïzena et les œufs, ajouter la levure et progressivement la farine jusqu'à obtention d'une pâte malléable.
3- Façonner des boules de pâte puis tremper l'emporte-pièce dans la farine.
4- Bien appuyer sur chaque boule pour obtenir la forme de rose. Décorer le centre avec un morceau de fruit confit puis enfourner.
5- Après cuisson, coller les gâteaux deux à deux avec de la confiture.
6- Badigeonner les bordures de confiture.
7- Enrober-les de noix de coco colorée.
plaisir de partage


gâteau invisible
POMME CHOCO-CARAMEL

Procedure


ALDEGHERI
ELECTRIC OVEN


gâteau invisible
POMME CHOCO-CARAMEL

Procedure


Galettes à la noix de coco
La pâte :
- 250 gr de beurre ramolli
- 1 verre de sucre glace
- 3 œufs
- 1 paquet de levure chimique
- 1 C. à café de vanille
- 1 verre de maïzena
- Faine (SIM)
La décoration :
- Fruits confits
- Noix de coco colorée
- Confiture
1- Préparer les ingrédients nécessaires.
2- Dans un récipient, mélanger le beurre, le sucre, la vanille, la maïzena et les œufs, ajouter la levure et progressivement la farine jusqu'à obtention d'une pâte malléable.
3- Façonner des boules de pâte puis tremper l'emporte-pièce dans la farine.
4- Bien appuyer sur chaque boule pour obtenir la forme de rose. Décorer le centre avec un morceau de fruit confit puis enfourner.
5- Après cuisson, coller les gâteaux deux à deux avec de la confiture.
6- Badigeonner les bordures de confiture.
7- Enrober-les de noix de coco colorée.

Procedure

In our context

- Data Science
- Data
- Data Scientist
- Algorithms: K-means, PageRank, etc.
- Databases
- Map-Reduce
- Hadoop, Spark
- Produce Value

Plan

Transaction

A transaction is an elementary unit of work carried out by a Database Management System application, to which we wish to allocate particular characteristics of reliability and isolation [1].

Terminology

The `COMMIT` command/action makes the changes made by a transaction permanents

After the commit all the changes made by the committed transaction become visible to others and are guaranteed to be durable

Transaction

ACID [5]

A transaction must possess the following properties

- Atomicity
  - Transaction is an indivisible unit of execution
- Consistency
  - Does not violate any of the integrity constraints defined on the database
- Isolation
  - The execution of a transaction is independent of the simultaneous execution of other transactions
- Durability
  - Demands that the effect of a transaction that has correctly executed a commit is not lost

Consistency

Consistency states that only valid data will be written to the database

If a transaction violates the databases consistency rules:

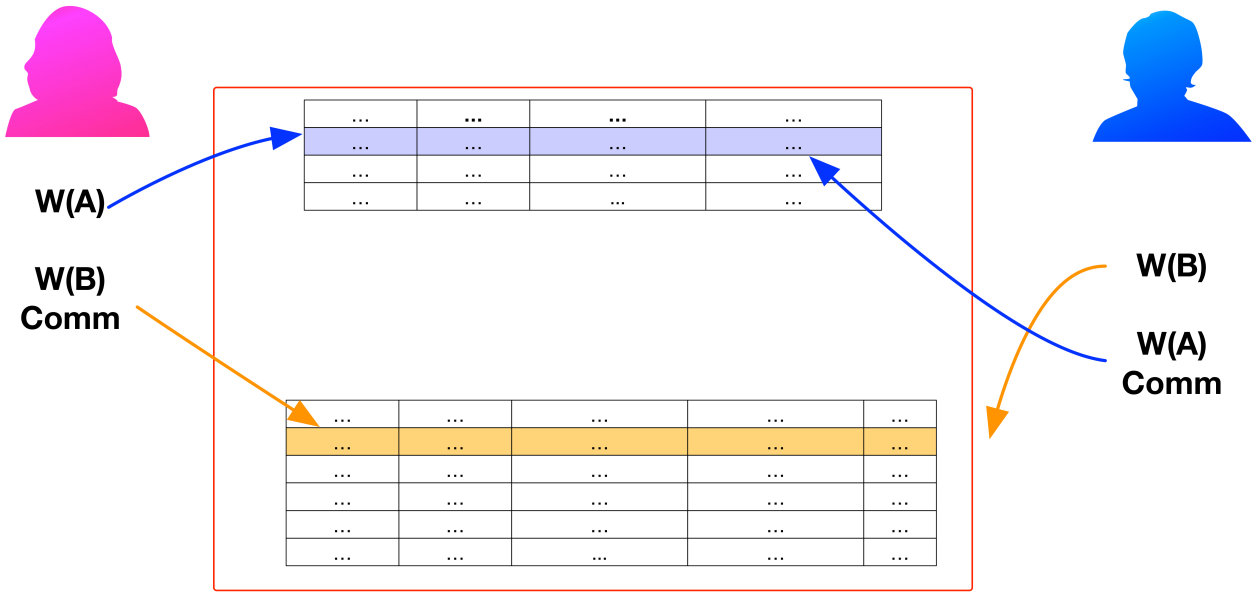
- The entire transaction is be rolled back
- The database is restored to a state consistent with those rules

Consistency

Concurrency

- write-write conflict
  - Two people updating the same data item at the same time
  - If the server serializes them: one is applied and immediately overwritten by the other (lost update)
- read-write conflict
  - Two serial inconsistent reads of the same data
- write-read conflict
  - A read after a possible not committed write
- read-read (or simply read) conflict
  - Different people see different data at the same time
  - Stale data: out of date

Write-write conflict

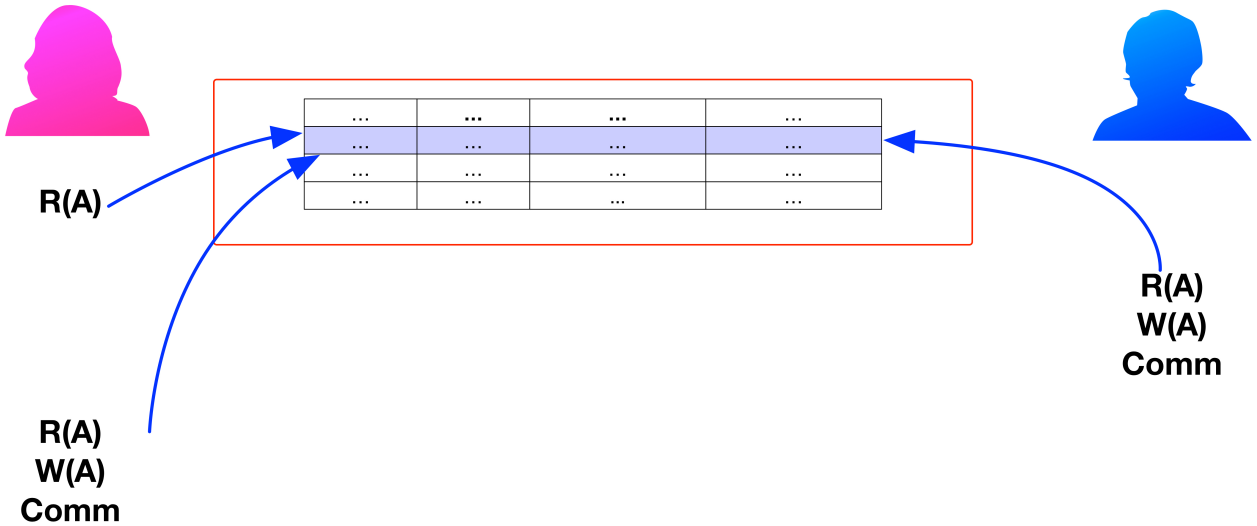


Overwriting uncommitted data

Two people updating the same data item at the same time

- There is no read: blind writes
- It is no possible to serialize this schedule

Read-write conflict

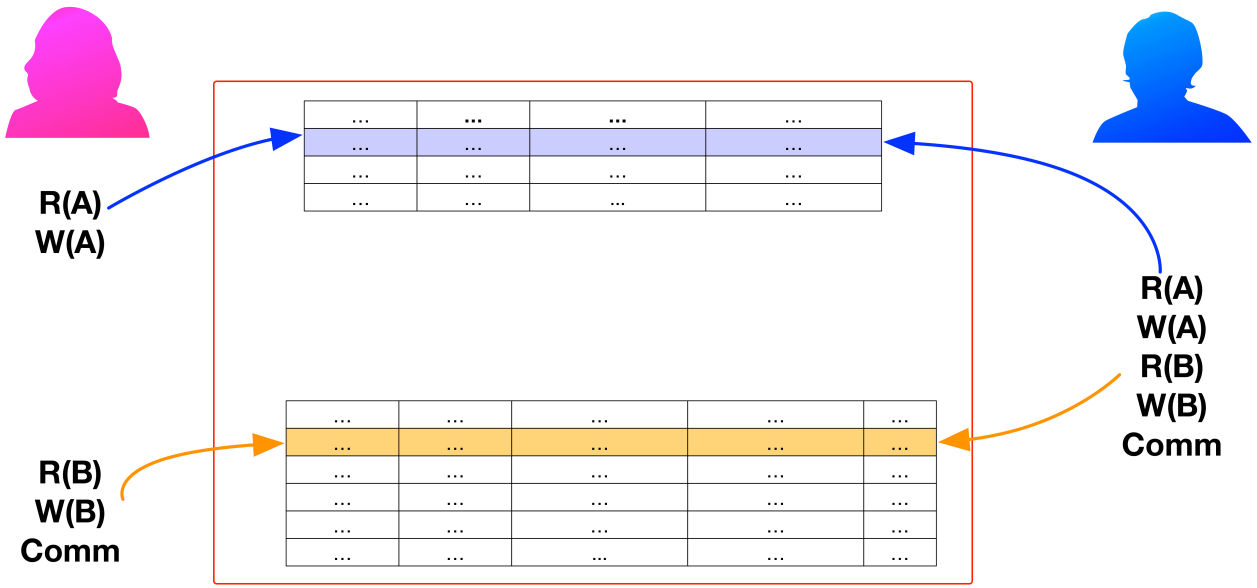


Unrepeatable read

Two serial inconsistent reads of the same data

- The value of A changes
- The first transaction is forced to abort

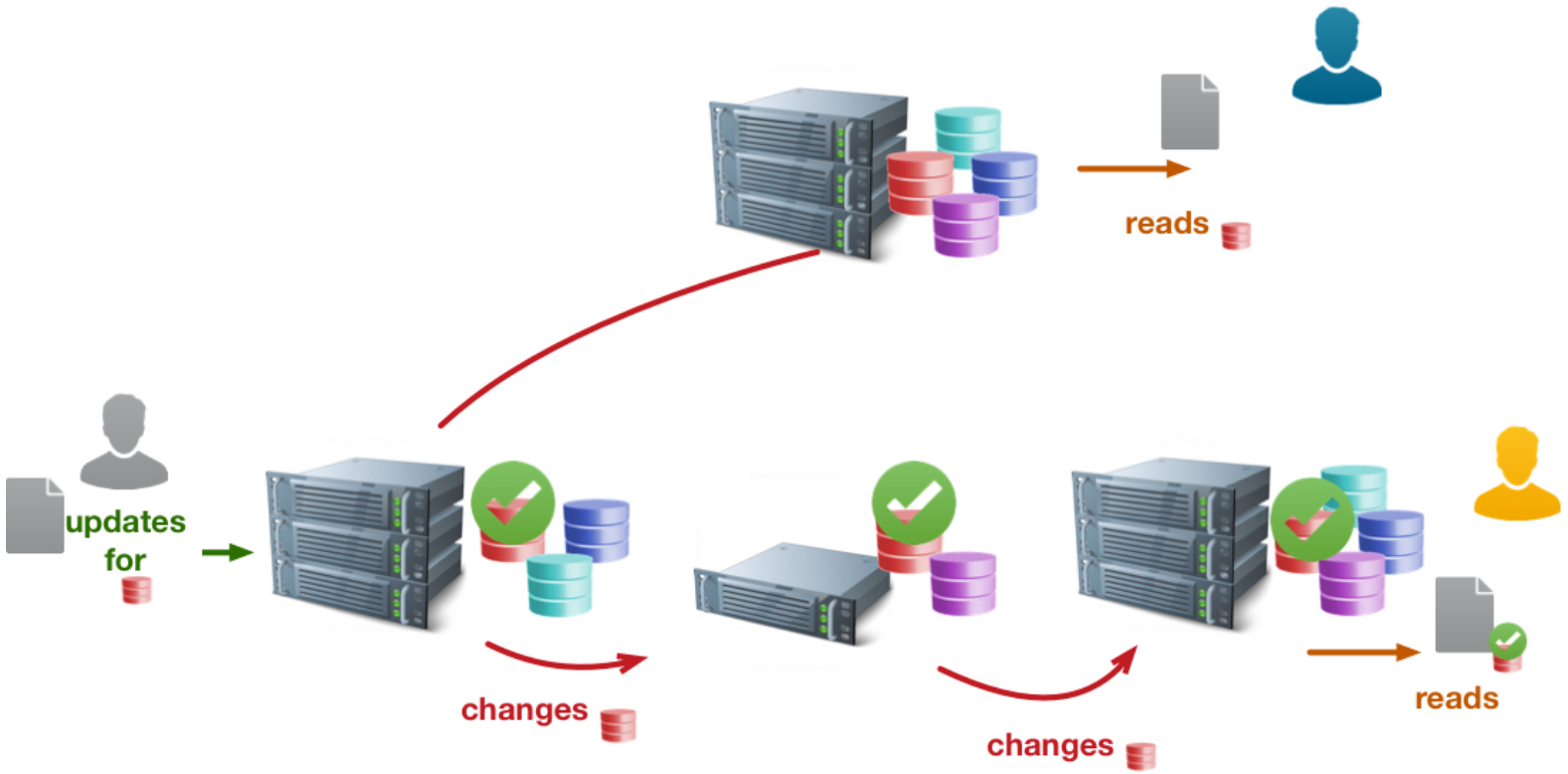
Write-read conflict



Reading uncommitted data

We have a dirty read: the second transaction is reading and overwriting data not yet committed by the first transaction

Read conflict



Replication is a source of inconsistency:

- Different data by different read actions

Solutions

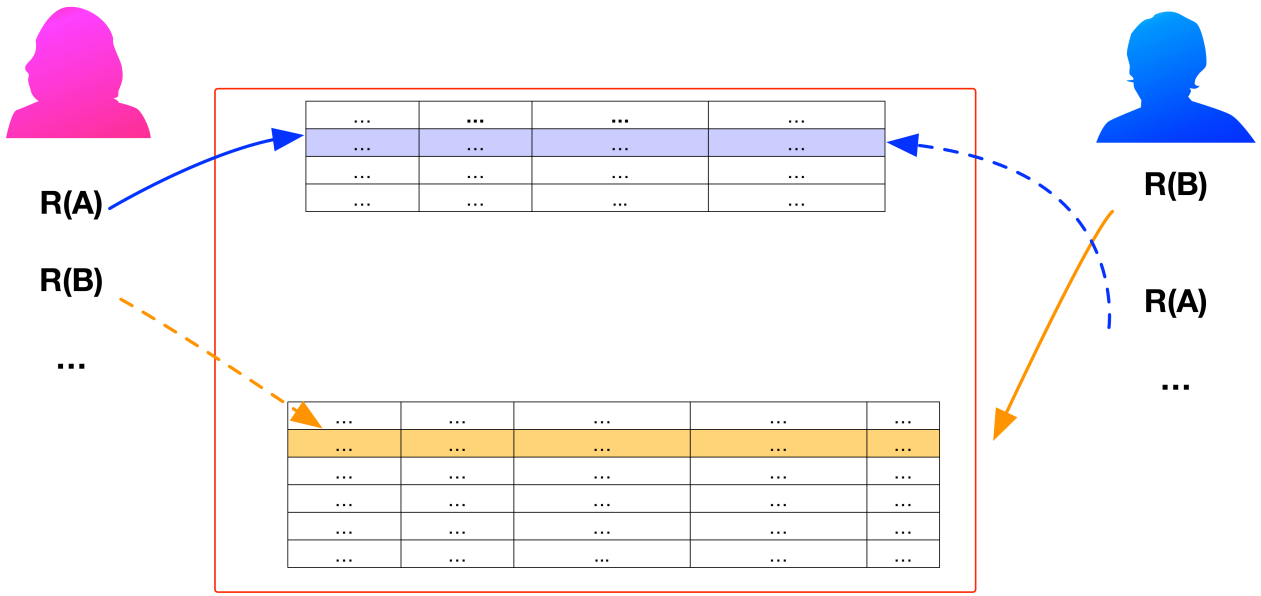
Pessimistic

- Prevent conflicts from occurring
- Implemented with write locks managed by the system

Relational Databases

Two-phase locking:

- A transaction acquires locks on the resources it reads/writes during the execution
- Another transaction cannot read/write the data it is working on
- Locks are released at the end of the transaction



Deadlocks

Result from the mutual blocking of two or more transactions

Solutions

Optimistic

- Lets conflicts occur, but detects them and takes action to sort them out
- Approaches:
  - Conditional updates: test the value just before updating
  - Save both updates: record that they are in conflict and then merge them

Analysis

Concurrency involves a fundamental tradeoff between:

- safety
  - avoiding errors such as update conflicts
- liveness
  - responding quickly to clients

Analysis

Concurrency involves a fundamental tradeoff between:

- safety
  - avoiding errors such as update conflicts
- liveness
  - responding quickly to clients

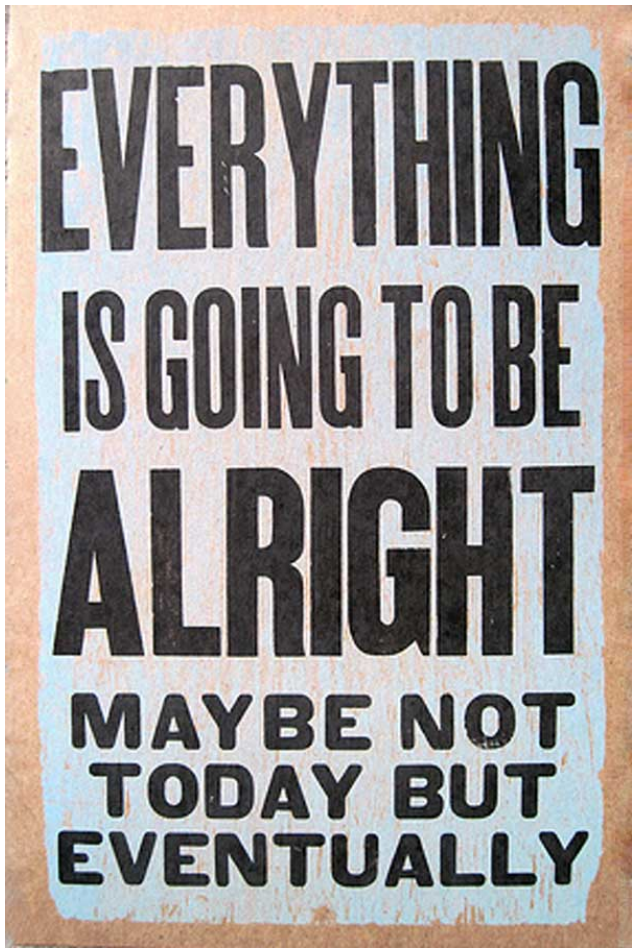
Pessimistic approaches often:

- severely degrade the responsiveness of a system
- lead to deadlocks, which are hard to prevent and debug

Forms of Consistency

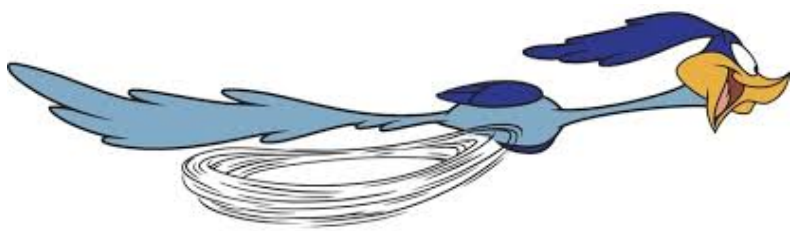
- Strong (or immediate) consistency
  - ACID transaction
- Logical consistency
  - No read-write conflicts (atomic transactions)
- Sequential consistency
  - Updates are serialized
- Session (or read-your-writes) consistency
  - Within a user's session
- Eventual consistency
  - You may have replication inconsistencies but eventually all nodes will be updated to the same value

Eventual Consistency



Transactions in NoSQL

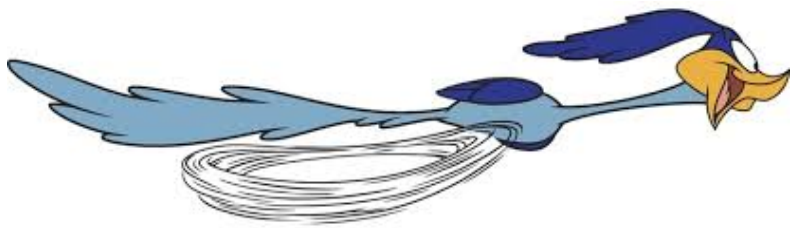
- Graph databases tend to support ACID transactions
- Aggregate-oriented NoSQL database:
  - Support atomic transactions: only within a single aggregate
  - Update over multiple aggregates: possible inconsistent reads
  - Inconsistency window: length of time an inconsistency is present
    - It will get there... eventually



Transactions in NoSQL

Amazon's documentation:

- Inconsistency window for SimpleDB service is usually less than a second ...



Transactions in NoSQL

Eventual consistency is independent from logical consistency but replication can lengthen the inconsistency window

- Two updates on the master performed in rapid succession
- Delays in networking can lengthen the inconsistency on a slave

Consequences of inconsistency windows:

- different people see different data at the same time: usually tolerated

Read-your-writes consistency: should be guaranteed

- Sticky session: tied to one node (session consistency)

Versions Stamps

Versions Stamp:

a value used to differentiate between different versions of a piece of data

- Help you detect concurrency conflicts
- When you read data, then update it, you can check the version stamp to ensure nobody updated the data between your read and write

Versions Stamps

- Version stamps can be implemented using:
  - counters
  - GUIDs
  - content hashes
  - current timestamps
  - a combination of these

With distributed systems a vector of version stamps (a set of counters, one for each node) allows you to detect when different nodes have conflicting updates

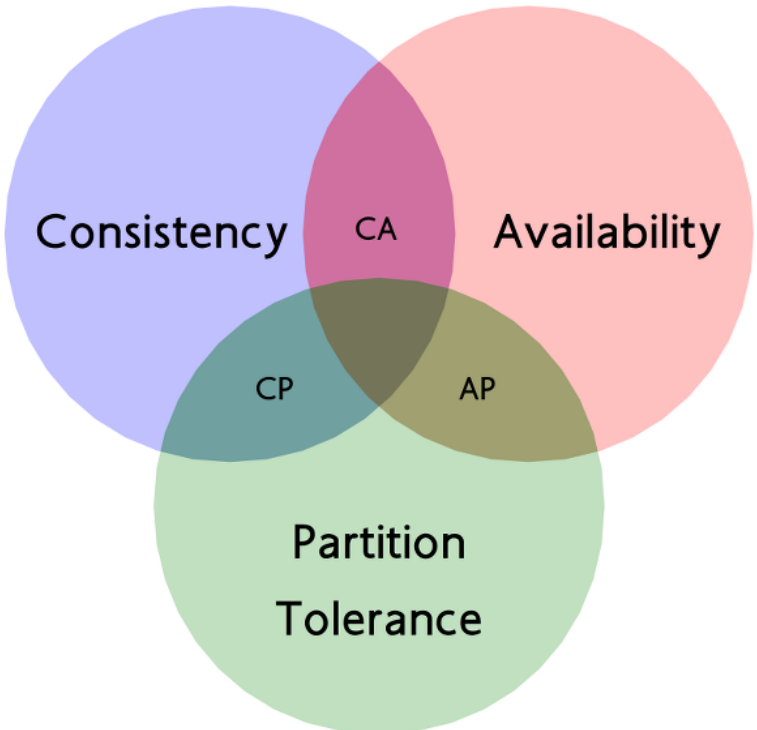
Plan

The CAP theorem

"Given the properties of Consistency, Availability, and Partition tolerance, you can only get two"

- Proposed by Eric Brewer in 2000 [3]
- Formal proof by Seth Gilbert and Nancy Lynch in 2002 [4]
- Sometimes it is necessary to relax consistency

Transactions in NoSQL



Maximize combinations of consistency and availability that make sense for the specific application [2]

CAP theorem

- Consistency: all people see the same data at the same time
- Availability: if you can talk to a node in the cluster, it can read and write data
- Partition tolerance: the cluster can survive communication breakages that separate the cluster into partitions unable to communicate with each other

CA Systems

- A single-server system is the obvious example of a CA system
  - Traditional RDBMS
- CA cluster: if a partition occurs, all the nodes would go down
  - Data center (rare and partial partitions)
- A system that suffers partitions
  - Distributed cluster
  - Tradeoff between consistency VS availability
  - Give up some consistency to get some availability

Example

- Bob is trying to book a room of the Ace Hotel in New York on a node located in London of a booking system
- Jim is trying to do the same on a node located in Mumbai
- The booking system uses a peer-to-peer distribution
- There is only one room available
- The network link breaks

Possible solutions

- CA: Neither user can book any hotel room
- CP: Designate Mumbai node as the master for Ace hotel
  - Jim can make the reservation
  - Bob can see the inconsistent room information
  - Bob cannot book the room
- AP: both nodes accept the hotel reservation
  - Overbooking!
- These situations are closely tied to the domain
  - Financial exchanges? Blogs? Shopping charts?
- Issues:
  - how tolerant you are of stale reads
  - how long the inconsistency window can be
- BASE approach (Basically Available, Soft state, Eventual consistency)

BASE approach

- Basic Availability
  - The system is going to respond to a request but the response can be a failure to obtain data, data in inconsistent state or in changing state
  - If a failure disrupts access to a segment of data, this does not necessarily result in a complete database outage
- Soft State
  - the state of the system is very likely to change over time
- Eventual consistency
  - The system will eventually become consistent once it stops receiving input: the data will propagate to everywhere that it should sooner or later go to
  - In reality the system will continue to receive input

Durability

- You may want to trade off durability for higher performance
  - Main memory database: if the server crashes, any updates since the last flush will be lost
  - Keeping user-session states as temporary information
  - Capturing sensor data from physical devices
- Replication durability: occurs when a master processes an update but fails before that update is replicated to the other nodes

Quorums

In this situation the more nodes are involved in a request, the higher is the chance of avoiding an inconsistency

Start to contact many nodes and check the returned values: trust the majority

- N: replication factor
- W: number of confirmed writes
- R: number of reads

Quorums

- How many nodes you need to contact to be sure you have the most up-to-date change?
  - Read quorum: `R + W > N`
- How many nodes need to be involved to get strong consistency?
  - Write quorum: `W > N/2`
- In a master-slave distribution one `W` and one `R` (to the master!) is enough
- A replication factor of 3 is usually enough to have good resilience

Summary I

- Write-write conflicts occur when two clients try to write the same data at the same time
- Read-write and Write-read conflicts occur when one client reads inconsistent data in the middle of another client's write
- Read conflicts occur when two clients read inconsistent data
- Pessimistic approaches lock data records to prevent conflicts
- Optimistic approaches detect conflicts and fix them
- Distributed systems (with replicas) see read(-write) conflicts due to some nodes having received updates while other nodes have not
- Eventual consistency means that at some point the system will become consistent once all the writes have propagated to all the nodes

Summary II

- Clients usually want read-your-writes consistency, which means a client can write and then immediately read the new value
- The CAP theorem states that if you get a network partition, you have to trade off availability of data versus consistency
- Durability can also be traded off against latency, particularly if you want to survive failures with replicated data
- It is not necessary to contact all replicas to preserve strong consistency with replication: a large enough quorum is sufficient

References I


P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone.
Databases: Concepts, Languages, Architectures.
McGraw-Hill, 1999.


E. Brewer.
Cap twelve years later: How the "rules" have changed.
Computer, 45(2):23–29, 2012.


Armando Fox and Eric A. Brewer.
Harvest, yield, and scalable tolerant systems.
In *Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, HOTOS '99, pages 174–, 1999.

References II

Seth Gilbert and Nancy Lynch.
Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.
SIGACT News, 33(2), June 2002.

Theo Haerder and Andreas Reuter.
Principles of transaction-oriented database recovery.
ACM Comput. Surv., 15(4):287–317, December 1983.