Large-Scale and Multi-Structured Databases

ACID vs BASE

Prof Pietro Ducange

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

IN SUPREMÆ DIGNITATIS 1343

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

The Tasks of DBMSs

Databases have to allow users to store and retrieve data. To this aim, three tasks must be in charge to DBMSs:

- Store data ***persistently***
- Maintain data ***consistency***
- Ensure data ***availability***

Distributed systems

Most of the recent NoSQL DBMSs can be deployed and used on ***distributed systems***, namely on ***multiple servers*** rather than a single machine.

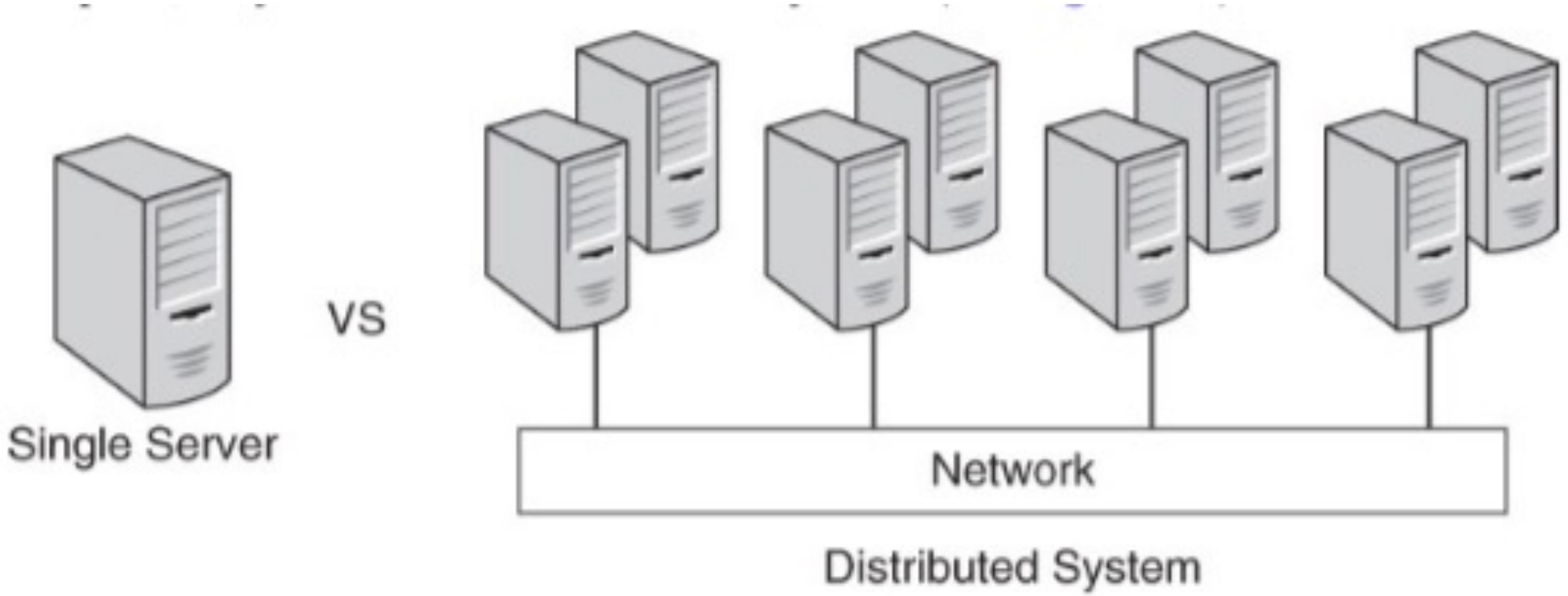


Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

Pros of Distributed Systems

- Ensure ***scalability***, ***flexibility, cost control*** and ***availability***
- It is easier to add or to remove nodes (***horizontal scalability***) rather than to add memory or to upgrade the CPUs of a single server (***vertical scalability***)
- Allow the implementation of ***fault tolerance*** strategies
- Accomplish with the ***motivations*** that led to the third databases revolution!

Some Cons of Distributed Systems

- To ***balance*** the requirements of data ***consistency*** and system ***availability***

- To protect themselves from ***network failures*** that may leave some nodes ***isolated***

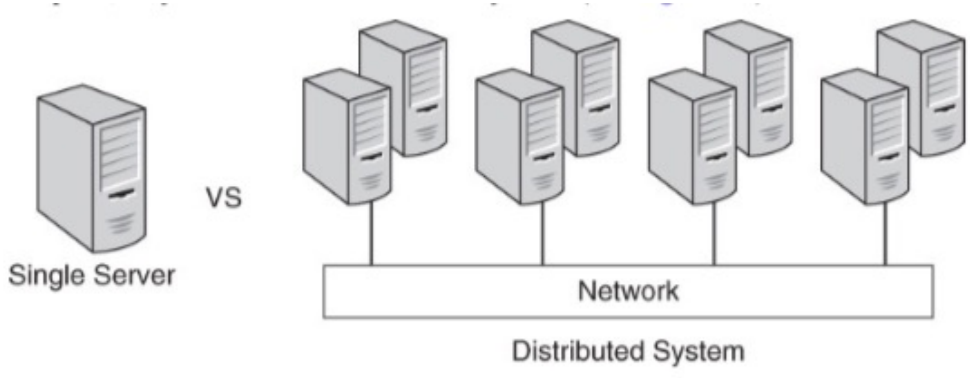


Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

Data Persistency

Data must be stored in a way that is ***not lost*** when the database server is shut down.

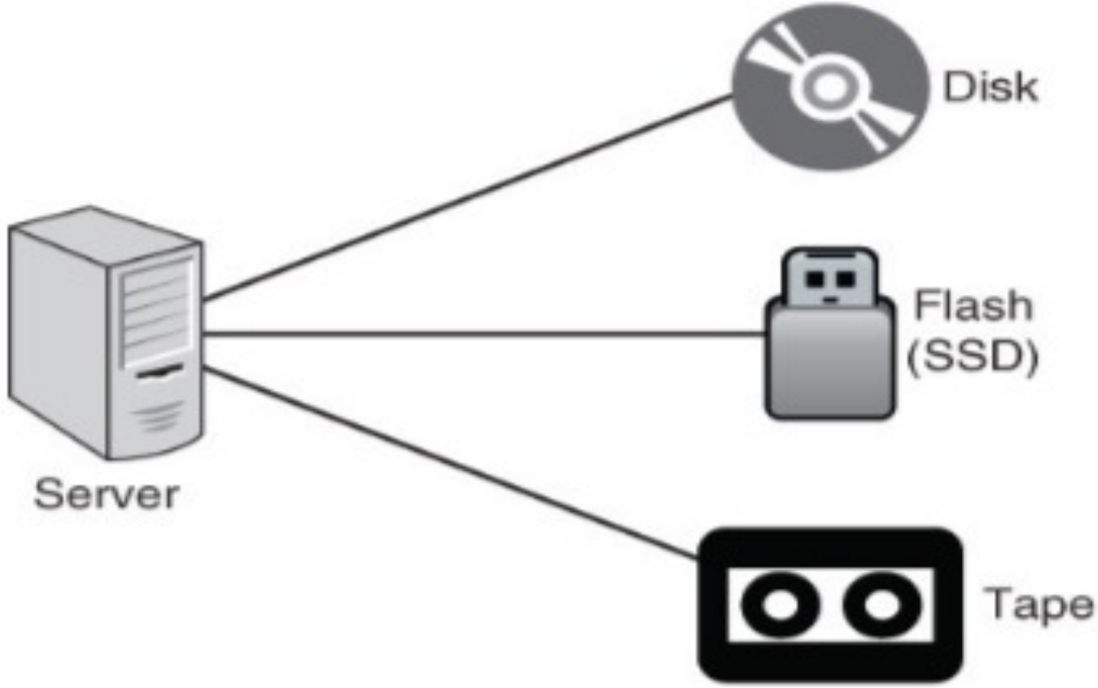
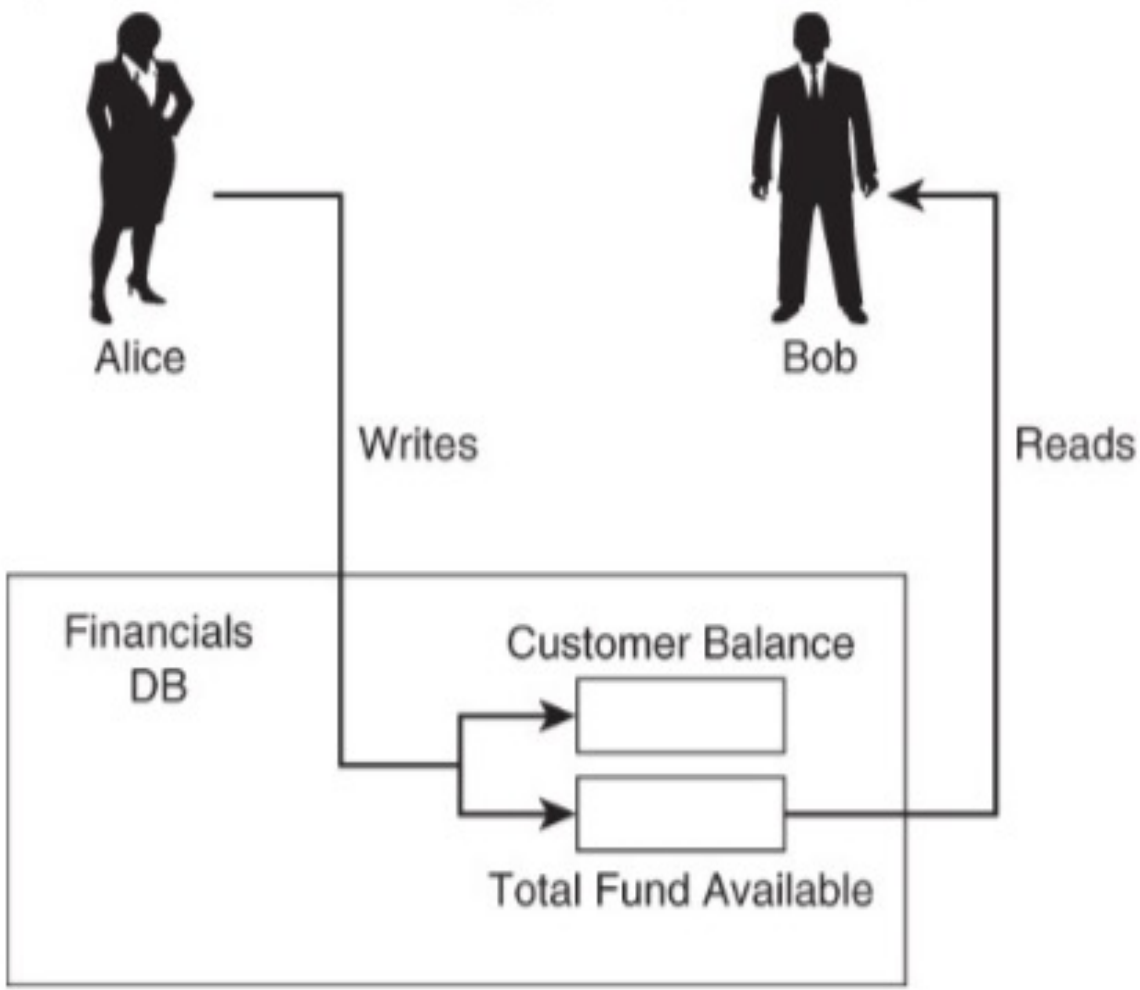


Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

Data Consistency and Integrity



Consistency and ***integrity*** of a database describe a state in which the stored data does not contradict itself.

Consistency ensures that a transaction can only bring the database from one ***valid*** state to another.

Integrity constraints are to ensure that data ***consistency*** is ***maintained*** for all insert and update operations.

Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

Data Availability

Data stored in a single server DBMS may be not available for several reasons, such as failures of the operating system, voltage drops, disks break down. In the figure we show a possible solution that ensures a high data availability: ***the two-phase commit.***

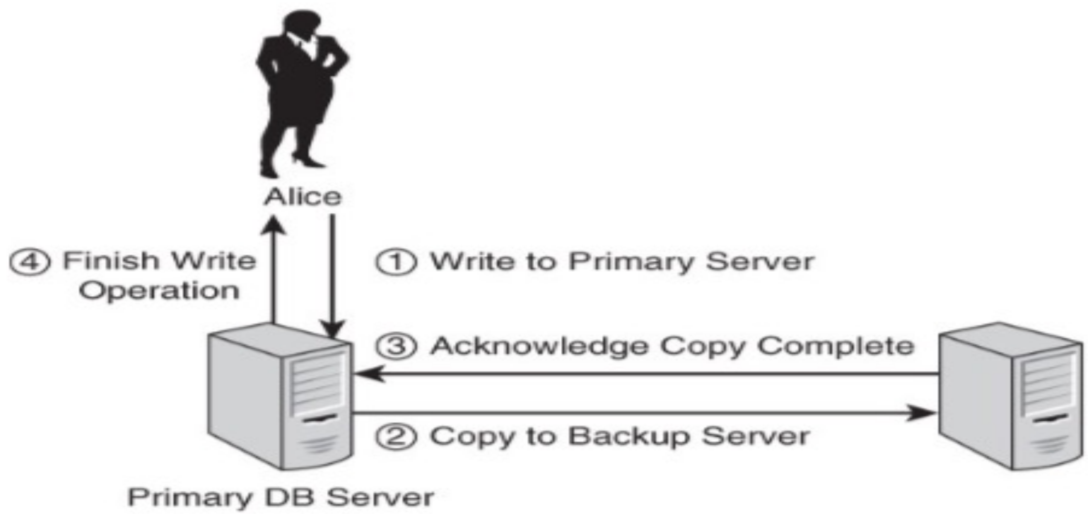


Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

If the primary server ***goes down***, the ***Backup server*** takes it places and the DBMS continues to offer its ***services*** to the users.

How long it takes the two-phase commit?

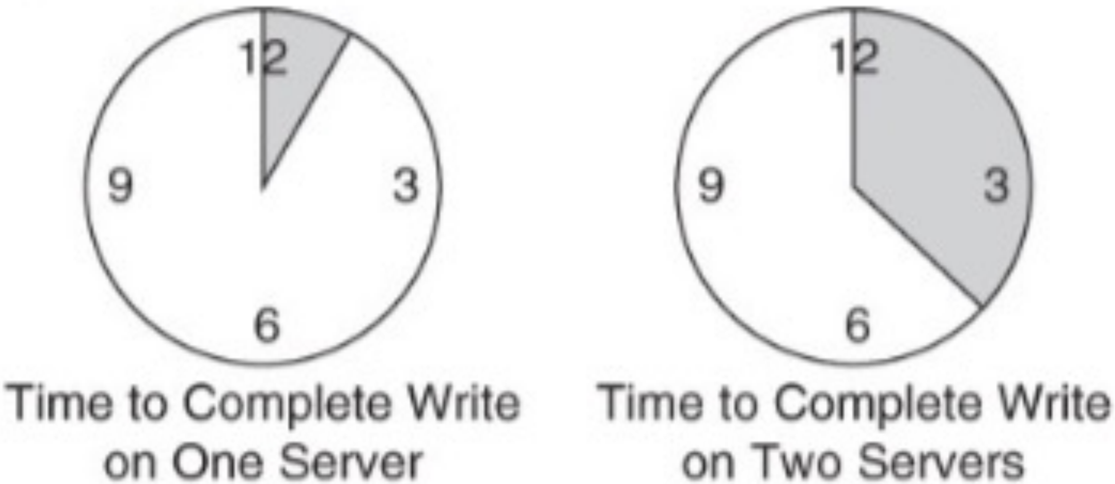


Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

Pay attention: the two-phase commit is a transaction!

You can have consistent data, but ***transactions will require longer times*** to execute than if you did not have those requirements.

ACID Transactions (I)

Jim Gray definition: “*A transaction is a transformation of state which has the properties of atomicity (all or nothing), durability (effects survive failures) and consistency (a correct transformation).*”

An ACID transaction should be:

Atomic: The transaction is ***indivisible***—either all the statements in the transaction are applied to the database or none are.

Consistent: The database remains in a consistent state ***before*** and ***after*** transaction execution.

ACID Transactions (II)

Isolated: While multiple transactions can be executed by one or more users simultaneously, one transaction should ***not see the effects*** of other in-progress transactions.

Durable: Once a transaction is saved to the database, its changes are expected to ***persist*** even if there is a ***failure*** of operating system or hardware.

TOO ACID!

There are some applications in which is not acceptable ***waiting too much*** time if we want concurrently consistent data and highly available systems.

In this kind of applications, the availability of the system and its ***fast response*** is more important than having consistent data on the different servers.

Practical example: ***e-commerce website***

Availability First!

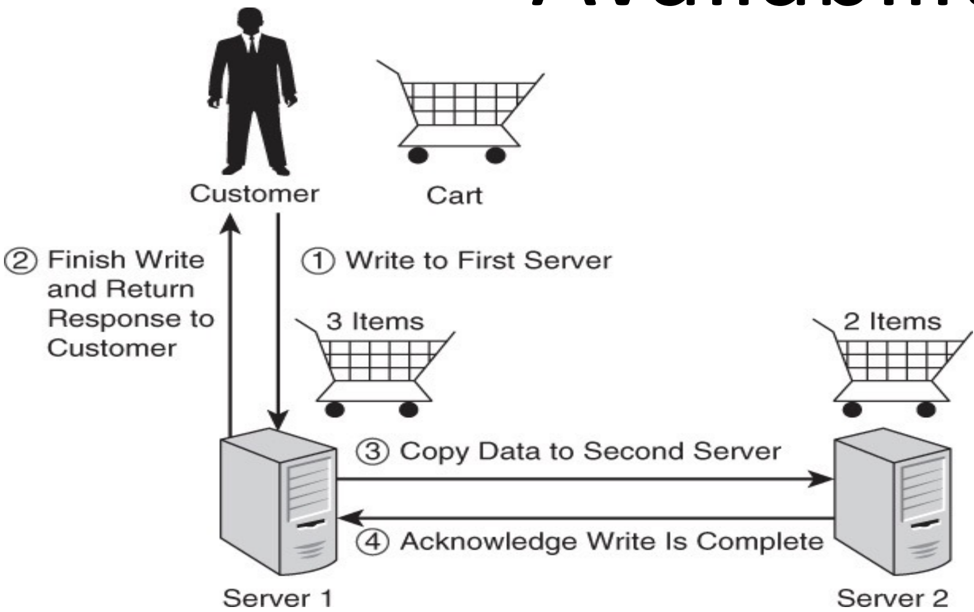


Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

NoSQL databases often implement ***eventual consistency***

There might be a period of time where copies of data have different values, ***but eventually all copies will have the same value.***

Amazon: even an "insignificant" tenth of a second in the response time of the site generates a reduction in sales estimated at around 1%;
Google: a measly half-second increase in latency caused by traffic can generate a 20% loss of requests.

Available but not Consistent

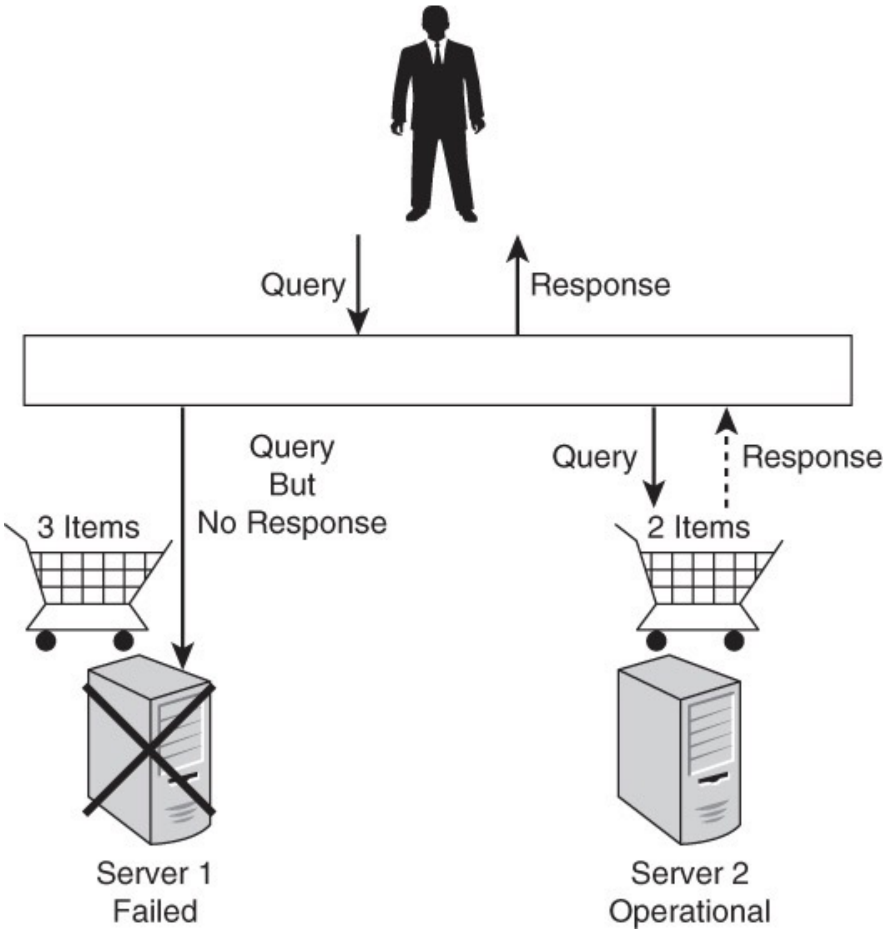


Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

In the e-commerce scenario the shopping cart may have ***a backup copy*** of the cart data that is out of sync with the primary copy.

The data would still be ***available*** if the primary server failed.

The data on the backup server would be ***inconsistent*** with data on the primary server if the primary server ***failed prior*** to updating the backup server.

Consistent but not Available

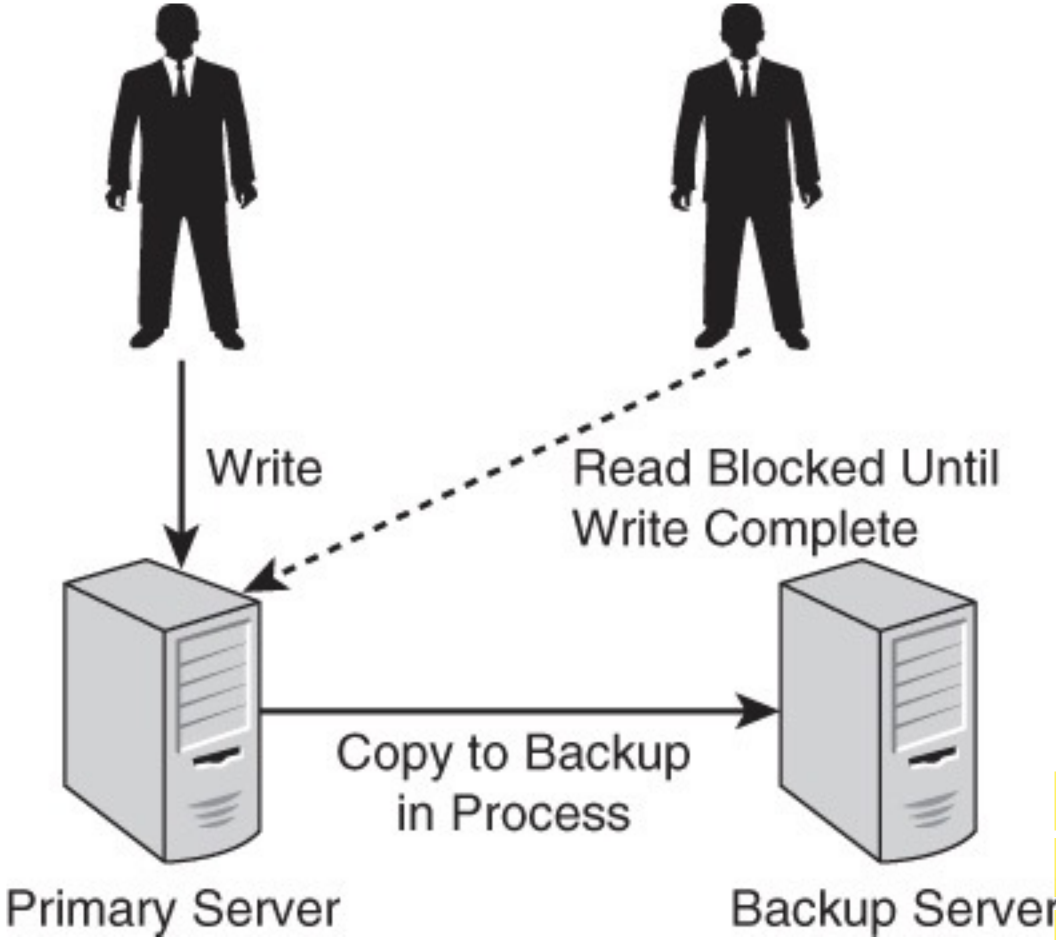


Image extracted from "Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015"

When dealing with ***two-phase commits*** we can have consistency but at the risk of the most recent ***data not being available*** for a brief period of time.

While the two-phase commit is executing, other ***queries*** to the data are ***blocked***.

The updated data is unavailable until the two-phase commit finishes. ***This favors consistency*** over availability

Network Partition

The system has ***two choices***: either show each user a different view of the data (availability but not consistency) or shut down one of the partitions and disconnect one of the users (consistency but not availability).

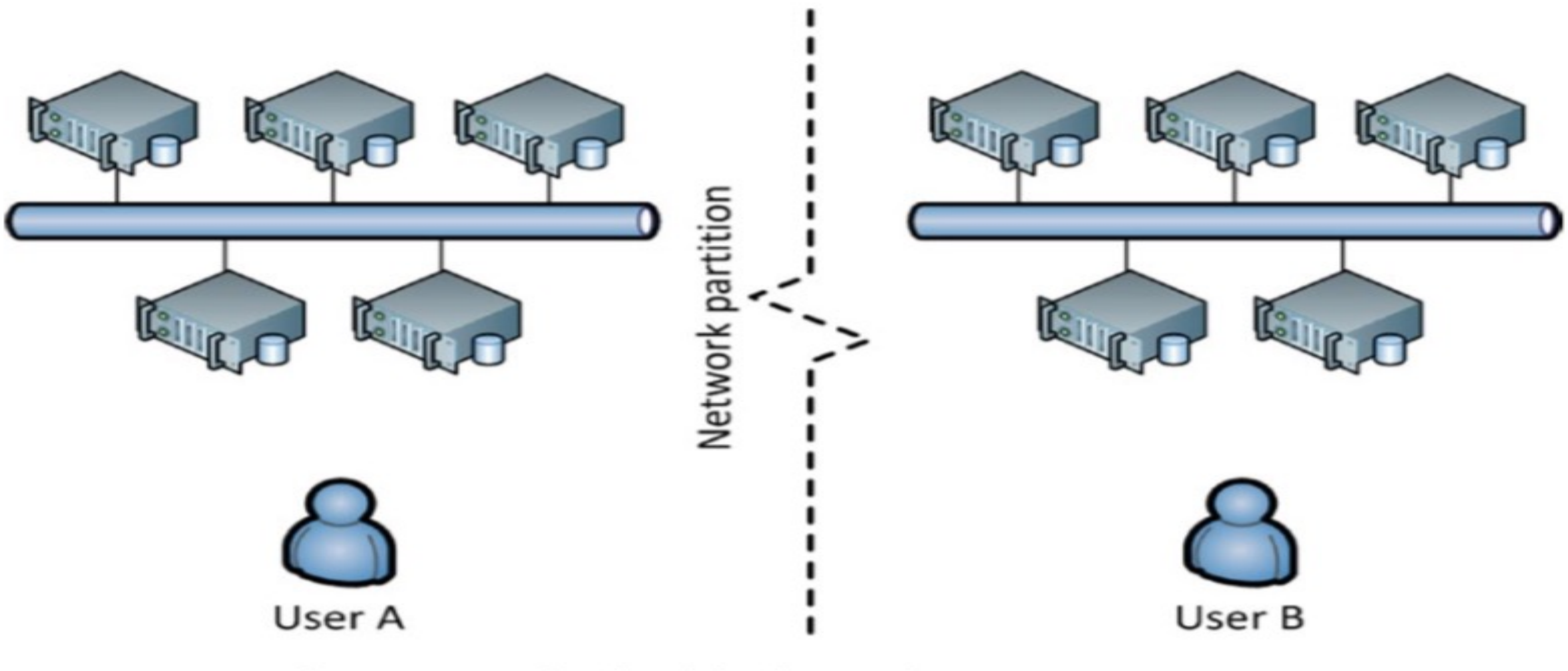


Image extracted from "Guy Harrison, Next Generation Databases, Apress, 2015"

The CAP Theorem (Brewer's theorem)

Distributed Databases cannot ensure at the same time:

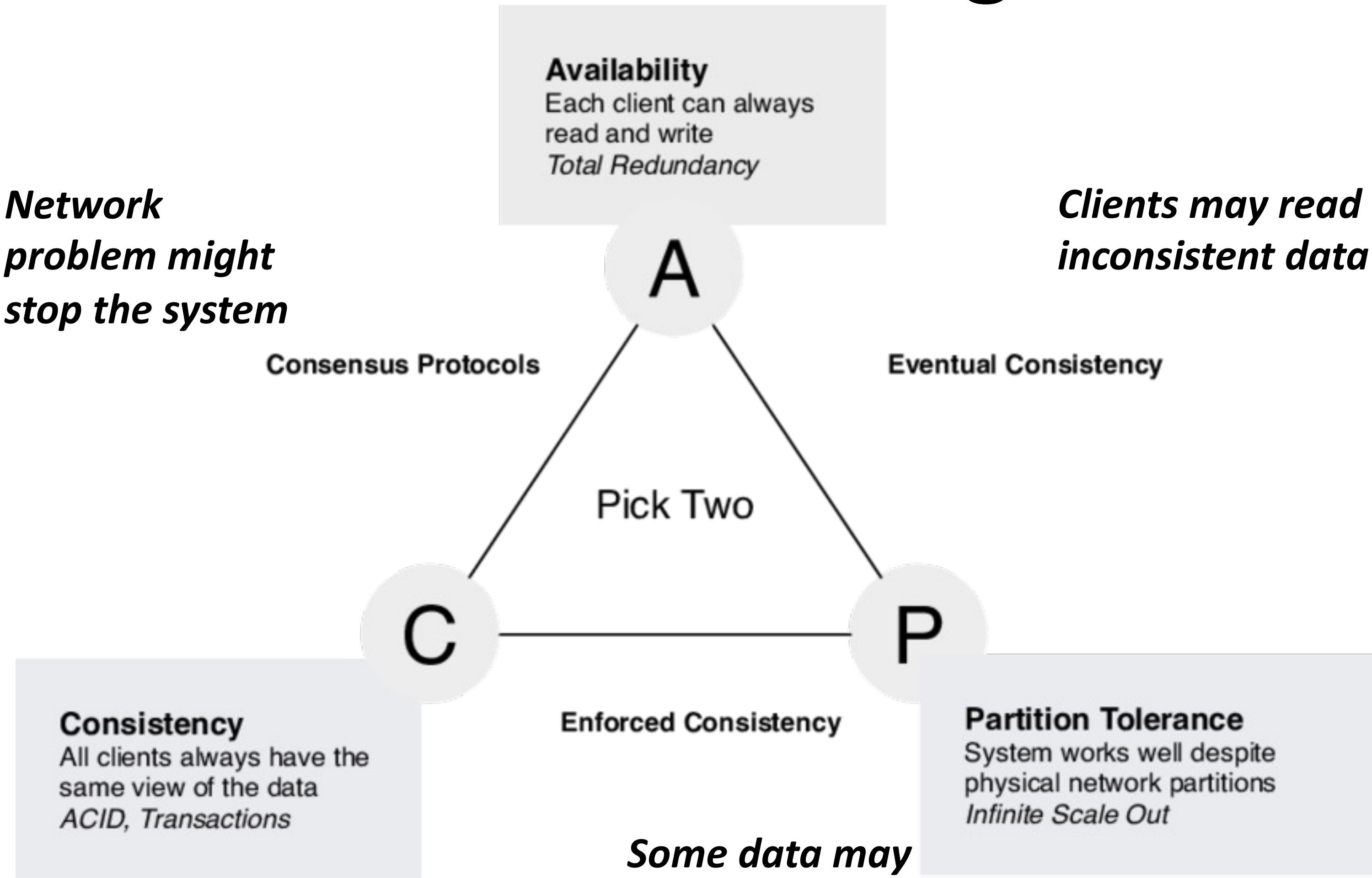
- ***Consistency (C)***, the presence of consistent copies of data on different servers
- ***Availability (A)***, namely to providing a response to any query
- ***Partition protection (P)***, Failures of individual nodes or connections between nodes do not impact the system as a whole.

At maximum ***two*** of the previous features may be found in a distributed database.

The CAP Triangle

Network problem might stop the system

Clients may read inconsistent data



Some data may become unavailable

Image extracted from: http://toppertips.com/cap-theorem/

CA Solutions

Single Site cluster, therefore all nodes are always in contact, when a partition occurs, the system blocks.

Choose C and A with compromising of P

Use cases: Banking and Finance application, system which must have transaction e.g. connected to RDBMS.

Total consistency can affect performance (latency).

AP Solutions

System is still available under ***partitioning***, but some of the data returned may be inaccurate.

Choose A and P with compromising of C.

Notice that this solution may return the ***most recent version*** of the data you have, which could be ***stale***. Indeed, this system state will also ***accept writes*** that can be processed later when the partition is resolved.

Availability is also a compelling option when the system needs to continue to function in spite of ***external errors***.
Use cases: shopping carts, News publishing CMS, etc.

CP Solutions

As suitable for application which require consistency, but also partition tolerance, while somewhat long response times are acceptable (***Bank ATMs***).

They are usually based on distributed and replicated relational or NoSQL systems supporting CP

In most of recent NoSQL frameworks the ***availability level*** can be setup with different parameters.

Choose based on the ***requirement analysis***.

BASE Properties of NoSQL Databases

- ***BA*** stands for ***basically available***: ***partial failures*** of the distributed database may be handled in order to ensure the ***availability*** of the service (often thanks to data ***replication***).

- ***S*** stands for ***soft state***: data stored in the nodes may be ***updated*** with ***more recent*** data because of the eventual consistency model (no user writes may be responsible of the updating!!).

- ***E*** stands for ***eventually consistent***: ***at some point in the future***, data in all nodes will converge to a consistent state.

The Latency Issue

High Availability

- High Availability is a strong requirement of modern shared-data systems

Replication

- To achieve High Availability, data and services must be replicated

Consistency

- Replication impose consistency maintenance

Latency

- Every form of consistency requires communication and a stronger consistency requires higher latency

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Types of Eventual Consistency

- ***Read-Your-Writes Consistency*** ensures that once a user has updated a record, all of his/her reads of that record will return the updated value.

- ***Session consistency*** ensures "read-your-writes consistency" during a session. If the user ends a session and starts another session with the same DBMS, there is no guarantee the server will "remember" the writes made by the user in the previous session.

- ***Monotonic read consistency*** ensures that if a user issues a query and sees a result, all the users will never see an earlier version of the value.

- ***Monotonic write consistency*** ensures that if a user makes several update commands, they will be executed in the order he/she issued them.

- If an operation ***causally depends*** on a preceding operation, there is a causal relationship between the operations. ***Causal consistency*** ensures that users will observe results that are consistent with the causal relationships.

Examples of Eventual Consistency (I)

Read-Your-Writes Consistency

Let's say Alice updates a customer's outstanding balance to $1,500.

The update is written to one server and the replication process begins updating other copies.

During the replication process, Alice queries the customer's balance.

She is guaranteed to see $1,500 when the database supports read- your-writes consistency.

Examples of Eventual Consistency (II)

Monotonic Read Consistency
Let's assume Alice is yet again updating a customer's outstanding balance.

The outstanding balance is currently $1,500.

She updates it to $2,500.

Bob queries the database for the customer's balance and sees that it is $2,500.

If Bob issues the query again, he will see the balance is $2,500 even if all the servers with copies of that customer's outstanding balance have not updated to the latest value.

Examples of Eventual Consistency (IIIa)

Monotonic Write Consistency (I)

Alice is feeling generous today and decides to reduce all customers' outstanding balances by 10%.

Charlie, one of her customers, has a $1,000 outstanding balance. After the reduction, Charlie would have a $900 balance.

Now imagine if Alice continues to process orders.

Charlie has just ordered $1,100 worth of material. His outstanding balance is now the sum of the previous outstanding balance ($900) and the amount of the new order ($1,100), namely $2,000.

Examples of Eventual Consistency (IIIb)

Monotonic Write Consistency (II)
Now consider what would happen if the database performed Alice's operations in a different order.

Charlie started with a \$1,000 outstanding balance.

Next, instead of having the discount applied, his record was first updated with the new order (\$1,100).

His outstanding balance becomes \$2,100.

Now, the 10% discount operation is executed and his outstanding balance is set to \$2,100– \$210, ***namely \$1890 (instead of \$2,000).***

Causal Consistency Example

Let suppose the following wall of a social network with comments and replies.

PIETRO: What a beautiful day!

    MIKE: Lucky you! Here it is raining cats and dogs!

LUCA: I passed the examination!

    MIKE: Great, you have to pay a beer to all the group tonight!

MIKE: check my last article on www.mike.org

time

Accepted View Order

MIKE: check my last article on www.mike.org

LUCA: I passed the examination!

    MIKE: Great, you have to pay a beer to all the group tonight!

PIETRO: What a beautiful day!

    MIKE: Lucky you! Here it is raining cats and dogs!

Not Accepted View Order

MIKE: check my last article on www.mike.org

    MIKE: Lucky you! Here it is raining cats and dogs!

    MIKE: Great, you have to pay a beer to all the group tonight!

PIETRO: What a beautiful day!

LUCA: I passed the examination!

Suggested Readings

Chapter 2 of the book “Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015”

Chapter 4 of the book ““Andreas Meier, Michael Kaufmann , SQL & NoSQL databases : models, languages, consistency options and architectures for big data management, 2019” https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45855.pdf

Brewer, Eric. "Pushing the cap: Strategies for consistency and availability." Computer 45.2 (2012): 23-29.

Chandra, Deka Ganesh. "BASE analysis of NoSQL database." Future Generation Computer Systems 52 (2015): 13-21.

http://sergeiturukin.com/2017/06/29/eventual-consistency.html