CSc 484
Database Management Systems

Ken Gamradt

Spring 2023

NoSQL Databases

Limitations of Relational Databases

- Relational databases have been the dominant type of database for decades
- Relational databases addressed many of the limitations of
  - Flat file–based data stores
  - Hierarchical databases
  - Network databases
- With the advent of the Web the limitations of relational databases became increasingly problematic
- Google, LinkedIn, Yahoo!, Amazon, … found that supporting large numbers of users on the Web was different from supporting much smaller numbers of business users, even those in large enterprises with thousands of users on a single database application

Limitations of Relational Databases

- Web application developers working with large volumes of data and extremely large numbers of users found they needed to support
  - Large volumes of read and write operations
  - Low latency response times
  - High availability
- These requirements were difficult to realize using relational databases
- These were not the first database users who needed to improve performance
- The problem is that techniques used in the past did not work at the scale of
  - Operations
  - Users
  - Data

  that businesses now demand

Limitations of Relational Databases

- In the past, if a relational database was running slowly, it could be upgraded with:
  - More CPUs
  - Additional memory
  - Faster storage devices
- This is a costly option and works only to a point
- There are limits to how many CPUs and how much memory can be supported in a single server
- Database designers could redesign the database schema to use techniques that would improve performance, but at the cost of increasing the risk of data anomalies – denormalization

Limitations of Relational Databases

- Another option is to use multiple servers with a relational database
- Operating a single relational database management system over multiple servers is a complex operation
- Long-term management becomes difficult
- There are also performance issues when supporting a series of operations that run on different servers, all must
  - complete successfully

  or
  - fail
- These sets of operations that succeed or fail together are known as transactions
- As the number of servers in a database cluster increases, the cost of implementing transactions increases

Motivations for Not Just/No SQL (NoSQL) Databases

- Pressing real-world problems motivated the data management professionals and software designers who created NoSQL databases
- Web applications serving tens of thousands or more users were difficult to implement with relational databases
- Four characteristics of data management systems that are particularly important for large-scale data management tasks are
  - Scalability
  - Cost
  - Flexibility
  - Availability
- Depending on the needs of a particular application, some of these characteristics may be more important than others

Scalability

- Ability to efficiently meet the needs for varying workloads
- E.g., if there is a spike in traffic to a website, additional servers can be brought online to handle the additional load
  - When the spike subsides, and traffic returns to normal
    - Some of those additional servers can be shut down
  - Adding servers as needed is called **scaling out**
- With relational databases, it is often challenging to scale out
- Additional database software may be needed to manage multiple servers working as a single database system
- E.g., Oracle offers Oracle Real Applications Clusters (RAC) for cluster-based databases
- Additional database components can add complexity and cost to operations

Scalability

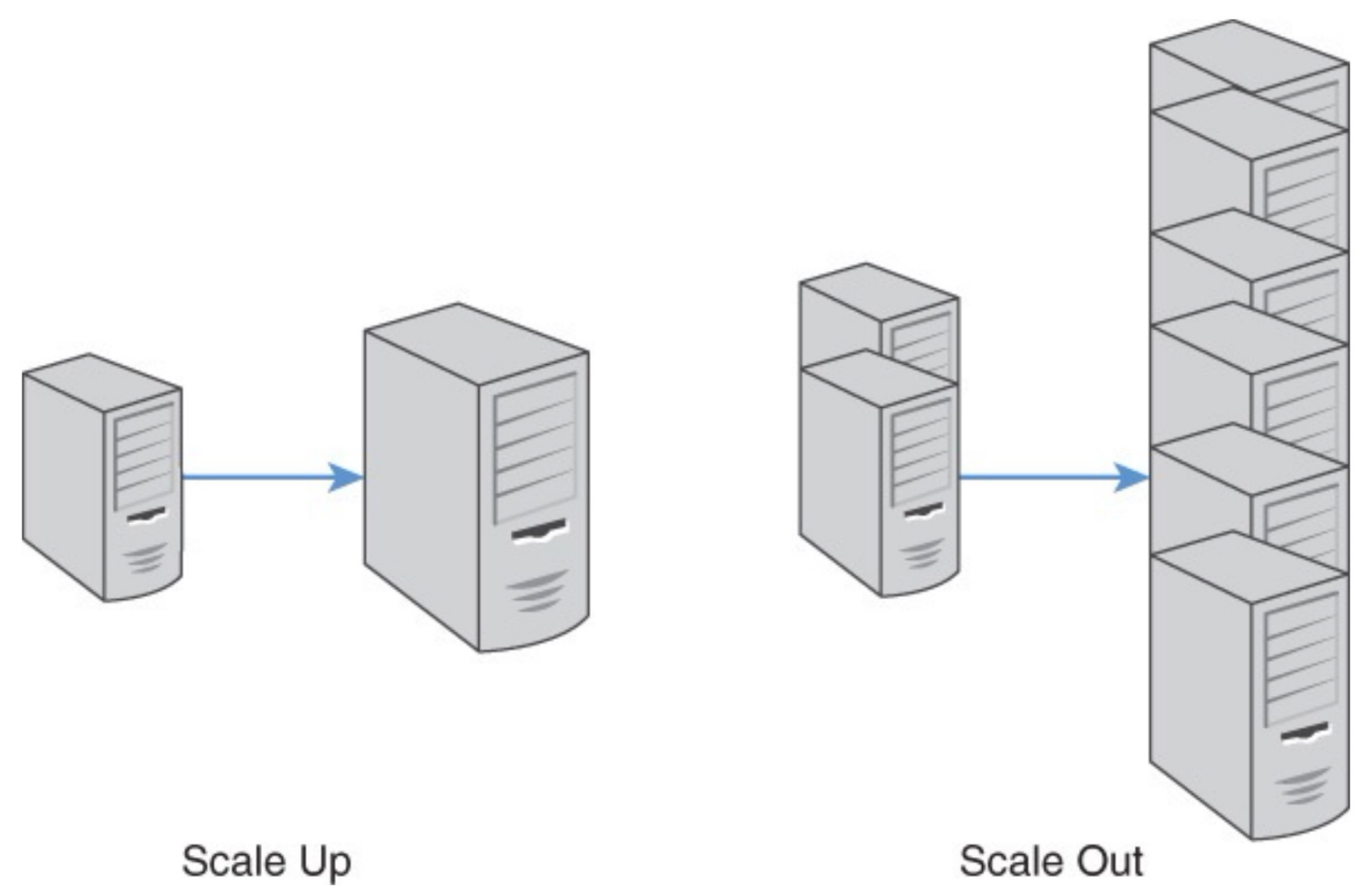
- Alternatively, database administrators could choose to **scale up**
- Upgrading an existing database server to add additional
  - Processors
  - Memory
  - network bandwidth
  - other resources

  that would improve performance on a database management system
- Replace an existing server with one with more
  - CPUs
  - Memory
  - …
- Scaling out is more flexible than scaling up
- Servers can be added or removed as needed when scaling up

Scalability

- NoSQL databases are designed to utilize servers available in a cluster with minimal intervention by database administrators
- As new servers are added or removed, the NoSQL database management system adjusts to use the new set of available servers
- Scaling up by replacing a server requires migrating the database management to a new server
- Scaling up by adding resources would not require a migration, but would likely require some downtime to add hardware to the database server

Scalability



Cost

- The cost of database licenses is an obvious consideration for any business or organization
- Commercial software vendors employ a variety of licensing models that include charging by the
  - size of the server running the RDBMS
  - number of concurrent users on the database
  - number of named users allowed to use the software

  Each of these models presents challenges for users of the database system

Cost

- Web applications may have spikes in demand that increase the number of users utilizing a database at any time
- Should users of the RDBMS pay for the
  - number of peak users
    - or
  - number of average users
- How should they budget for RDBMS licenses when it is difficult to know how many users will be using the system six months or a year from now?

Cost

- Users of open source software avoid these issues
- The software is free to use on as many servers of whatever size needed because open source developers typically do not charge fees to run their software
- Fortunately for NoSQL database users, the major NoSQL databases are available as open source
- Third-party companies provide commercial support services for open source NoSQL databases so businesses can have software support as they do with commercial relational databases

Flexibility

- Relational database management systems are flexible in the range of problems that can be addressed using relational data models
- Industries as different as
  - Banking
  - Manufacturing
  - Retail
  - Energy
  - Health care

  all make use of relational databases

Flexibility

- There is, however, another aspect of relational databases that is less flexible
- Database designers expect to know at the start of a project all the tables and columns that will be needed to support an application
- It is also commonly assumed that most of the columns in a table will be needed by most of the rows
- E.g., all employees will have names and employee IDs
- There are times that the problems modeled are less homogeneous than that

Flexibility

- Consider an e-commerce application that uses a database to track attributes of products
  - Computer products would have attributes such as
    - CPU type
    - Amount of memory
    - Disk size
  - Microwave ovens would have attributes such as
    - Size
    - Power
- A database designer could create separate tables for each type of product or define a table with as many different product attributes as they could imagine at the time the database is designed

Flexibility

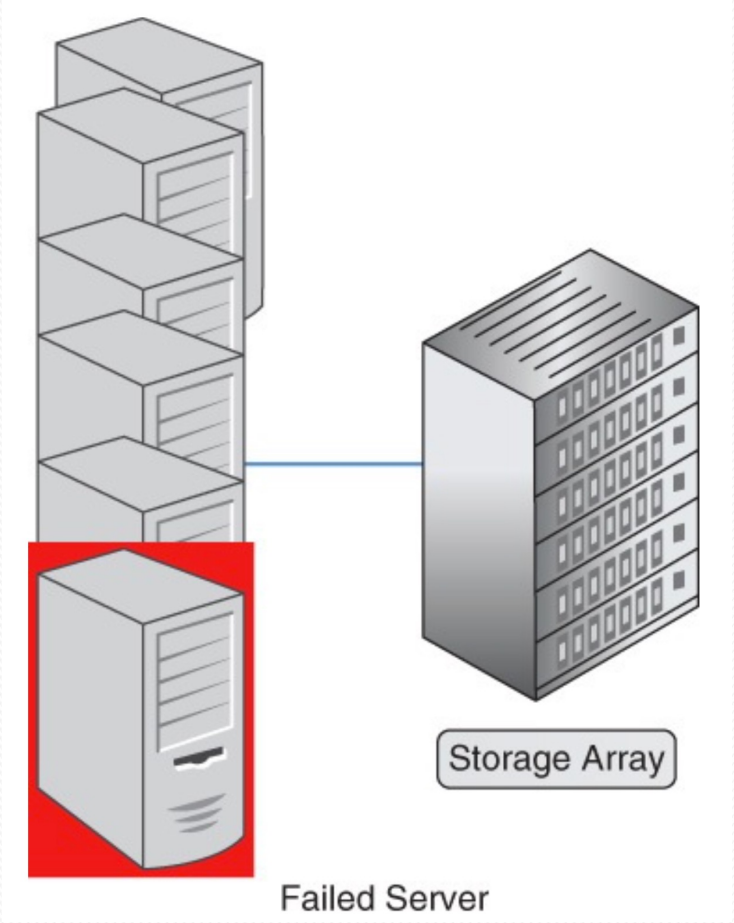
- Unlike relational databases, some NoSQL databases do **NOT** require a fixed table structure
- E.g., in a document database, a program could dynamically add new attributes as needed without having to have a database designer alter the database design

Availability

- Many of us have come to expect websites and web applications to be available whenever we want to use them
- If our favorite social media or e-commerce site were frequently down when we tried to use it, we would likely start looking for a new favorite
- NoSQL databases are designed to take advantage of
  - multiple
  - low-cost

  servers
- When one server fails or is taken out of service for maintenance, the other servers in the cluster can take on the entire workload
- Performance may be somewhat less, but the application will still be available

Availability

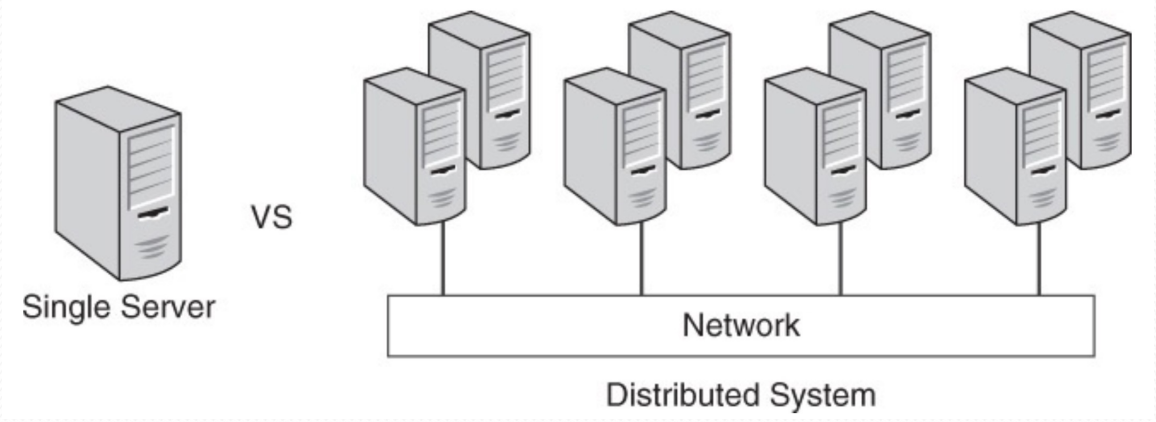


Availability

- If a database is running on a single server and it fails, the application will become unavailable unless there is a backup server
- Backup servers keep replicated copies of data from the primary server in case the primary server fails
- If that happens, the backup can take on the workload that the primary server had been processing
- This can be an inefficient configuration because a server is kept in reserve in the event of a failure but otherwise it is not helping to process the workload

Variety of NoSQL Databases

- NoSQL databases solve a wide variety of data management problems by offering several types of solutions
- NoSQL databases are commonly designed to use multiple servers
  - This is not a strict requirement
- When systems run on multiple servers they are known as **distributed systems**



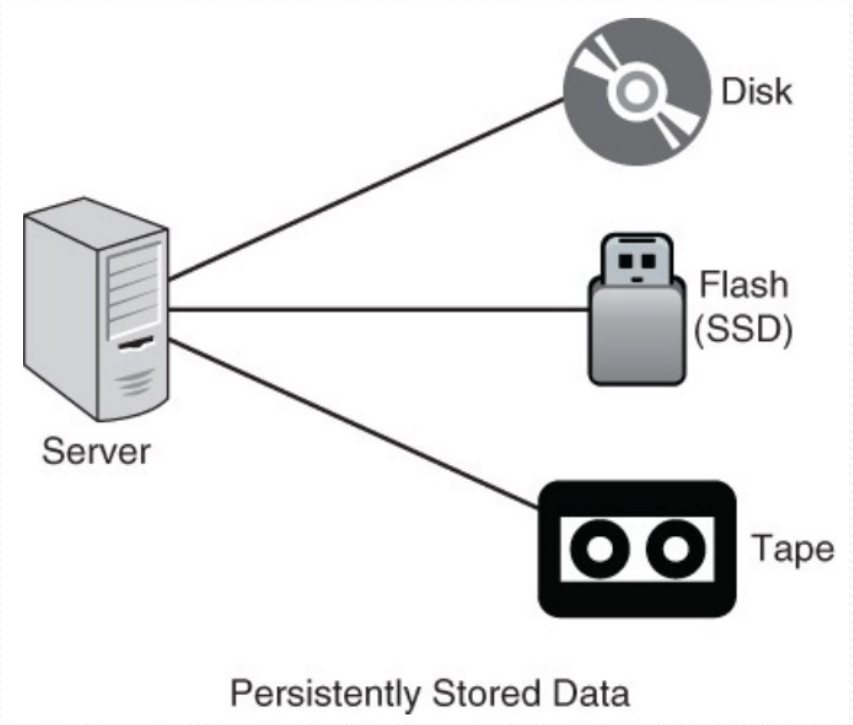
Data Management with Distributed Databases

- Let's look at a simplified view of databases in general
- Databases are designed to do two things:
  - store data
  - retrieve data
- The database management systems must do three things:
  - Store data persistently
  - Maintain data consistency
  - Ensure data availability

Store Data Persistently

- Data must be stored persistently
  - Stored in a way that data is not lost when the database server is shut down
- If data were only stored in memory, RAM, then it would be lost when power to the memory is lost
- Only data that is stored on
  - Disk
  - Flash
  - Tape
  - other long-term storage

  is considered persistently stored



Store Data Persistently

- Data must be available for retrieving
- We can retrieve persistently stored data in several different ways
  - Data stored on a flash device is read directly from its storage location
  - The movable parts of the disk and tape drives are put in position so that the read heads of the device are over the block of data to be read
- We could design our database to simply start at the beginning of a data file and search for the record we need when a read operation is performed
- This would lead to painfully long response times and a waste of valuable compute resources
- Rather than scan the full table for the data, we can use database indexes, which are like indexes at the end of a book, to quickly find the location of a particular piece of data

Maintain Data Consistency

- It is important to ensure that the correct data is written to a persistent storage device
- If the write or read operation does not accurately record or retrieve data, the database will not be of much use
- This is rarely a problem unless there is a hardware failure
- A more common issue with reading and writing occurs when two or more people are using the database and want to use the same data at the same time

Maintain Data Consistency

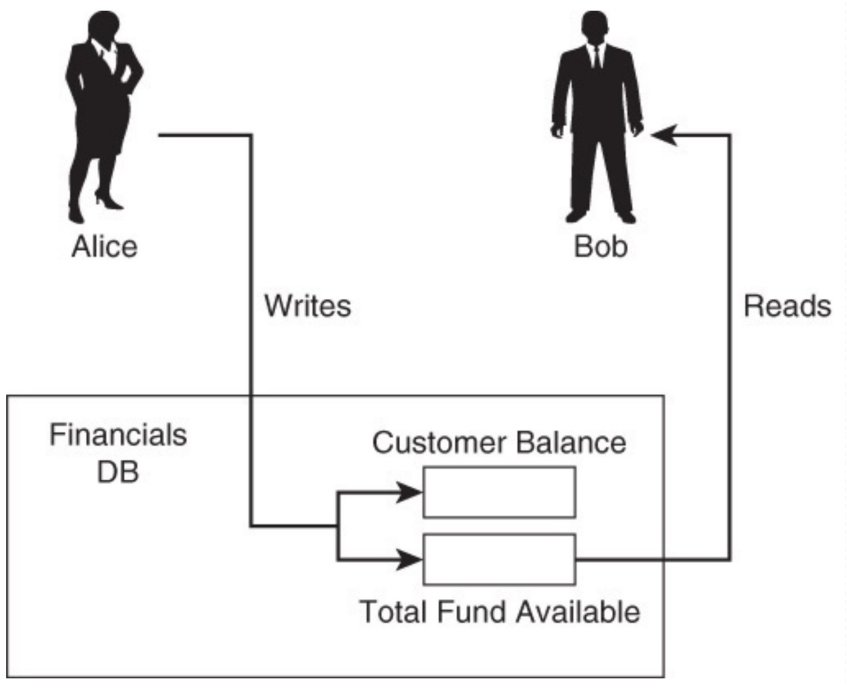
- Consider a small business with two partners, Alice and Bob
- Alice is using a database application to update the company's financial records
- She has just received several payments from customers and posted them to the accounting system
- The process requires two steps updating the:
  - customer's balance due
  - updating the total funds available to the business
- At the same, Bob is placing an order for more supplies
- Bob wants to make sure there are sufficient funds available before he commits to an order, so he checks the balance of total funds available
- What balance will Bob see?

Maintain Data Consistency

- Ideally, Bob would see the balance of funds available that includes the most recent payments
- If he issues his query while Alice is updating the customer balance and total funds available, then Bob would see the balance without the new payments
- The reason for this is that the database is designed to be consistent
- Bob could see the balance before or after Alice updates both the
  - customer balance record
  - funds available record

  but never when only one of the two has been updated

Maintain Data Consistency



- It would be inconsistent for the database to return results that indicated a customer had paid the balance due on their account without also including that payment in the funds available record
- Relational database systems are designed to support these kinds of multistep procedures that must be treated as a single operation or transaction

Ensuring Data Availability

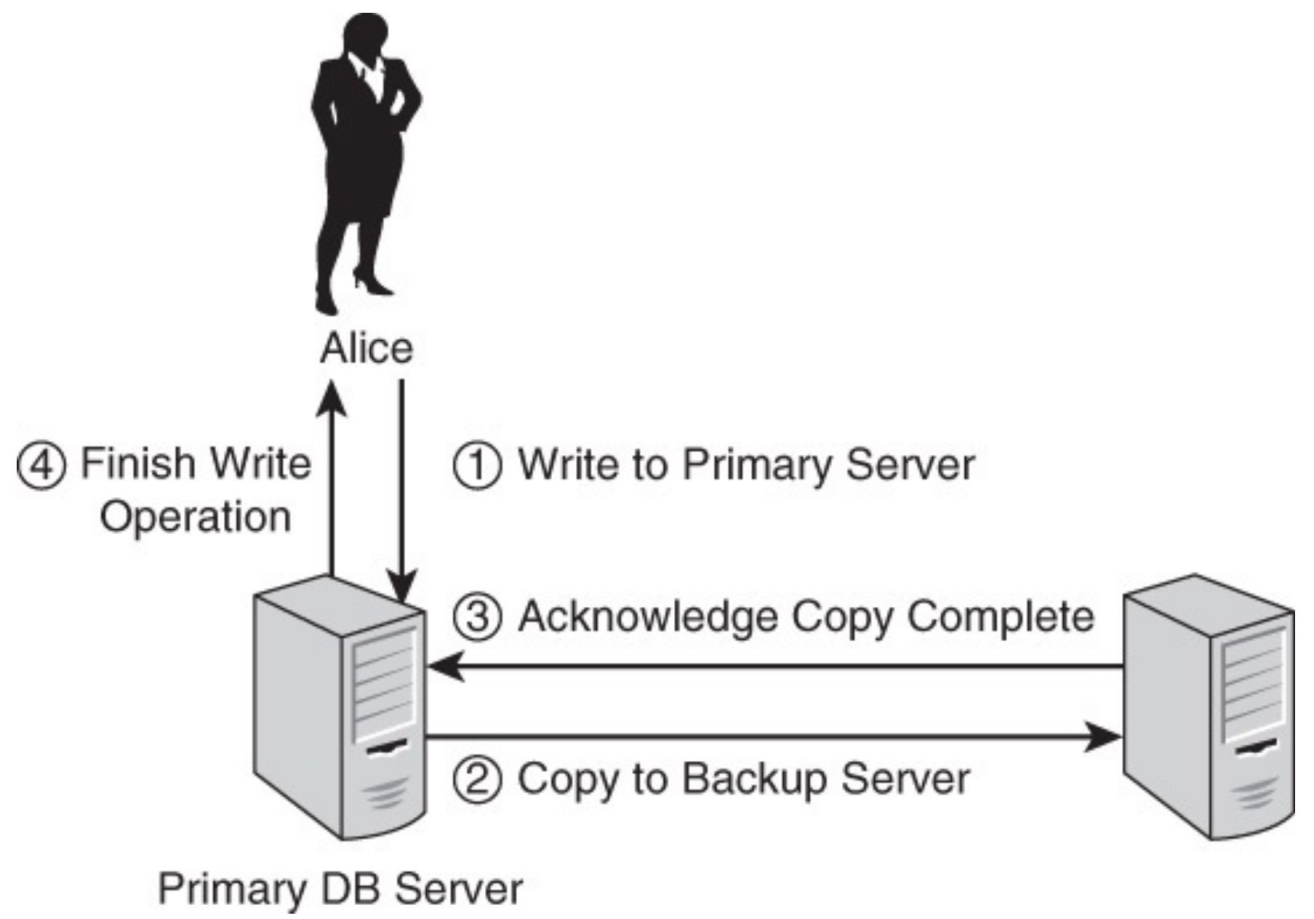
- Data should be available whenever it is needed – difficult to guarantee
  - Hardware may fail
  - Operating system on the database server may need patching
  - Might need to install a new version of the database management system
  - Database that runs on a single server can be unavailable for many reasons
- One way to avoid the problem of an unavailable database server is to have two database servers:
  - One is used for updating data and responding to queries – primary server
  - One is kept as a backup in case the first server fails – backup server
- The backup server starts with a copy of the database on the primary server
- When changes are made to the primary database, they are reflected on the backup database as well

Ensuring Data Availability

- E.g., if Alice and Bob's company used a backup database server, then when Alice updated a customer's account, those same changes would be made to the backup server
- This would require the database to write data twice:
  - once to the disk used by the primary server
  - once to the disk used by the backup server

  in an operation known as a two-phase commit

Ensuring Data Availability



Ensuring Data Availability

- A database transaction is an operation made up of multiple steps and that all steps must complete for the transaction to complete
- If any one of the multiple steps fails, then the entire transaction fails
- Updating two databases makes every update a multistep process
- When the company used a single server, there was just one step in updating the number of a particular product in the warehouse
- E.g., the number of black desk chairs could be updated from 100 to 125 in a single operation
- Now that the company is using a backup database, the number of chairs would have to be updated on the primary server and the backup server

Ensuring Data Availability

- The process for updating both databases is similar to other multistep transactions:
  - Both databases must succeed for the operation to succeed
- If the primary database is updated to reflect 125 black desk chairs in the warehouse but the update fails on the backup database, then the primary database resets (rolls back) the chair count back to 100
- The primary and the backup databases must be consistent
- This is an example of a two-phase commit
  - In the first phase of the operation, the database writes, or commits, the data to the disk of the primary server
  - In the second phase of the operation, the database writes data to the disk of the backup server

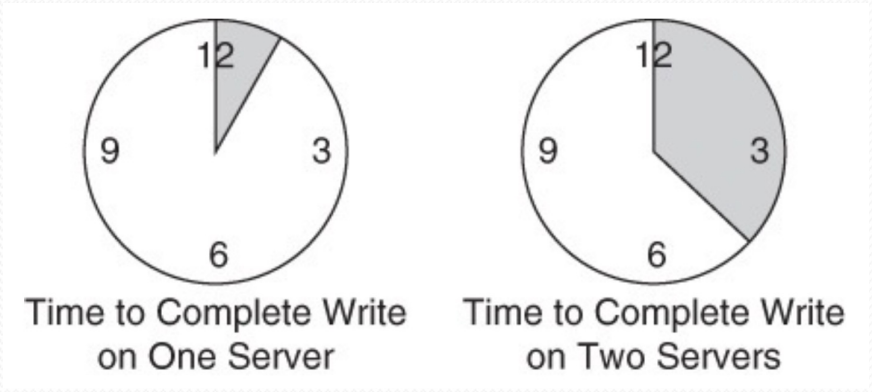
Ensuring Data Availability

- With data consistent on two database servers, we can be sure that if the primary database fails, we can safely switch to using the backup database
- When the primary database is back online, the first thing it does is to update itself so that all changes made to the backup database are made to the primary database
  - The primary database is usable when it is consistent with the backup database
- The advantage of using two database servers is that it enables the database to remain available even if one of the servers fails
- This is helpful but is not without costs
  - database applications
  - people who use them

  must wait while a write operation completes

Ensuring Data Availability

- In the case of a two-phase commit, a write operation is not complete until both databases are updated successfully
- The speed of the updates depends on the
  - amount of data written
  - speed of the disks
  - speed of the network between the two servers
  - other design factors



Consistency of Database Transactions

- The term consistency, with respect to database transactions, refers to maintaining a single, logically coherent view of data
- When we transfer $100 from our savings account to our checking account, the bank's software may
  - subtract $100 from our savings account in one step
  - add $100 to our checking account in another
- At no time would it be correct to say we have $100 less in our savings account without also reflecting an additional $100 in our checking account
- Consistency has also been used to describe the state of copies of data in distributed systems
- E.g., if two database servers each have copies of product data stored in a warehouse, it is said they are consistent if they have the same data
  - This is different from the kind of consistency that is needed when updating data in a transaction

Consistency of Database Transactions

Note

To avoid confusion going forward, let's define a database server as a computer that runs database management software. That database management software will be called a database management system.

Database management systems can run on one or more computers. When the database management system is running on multiple computers, it is called a distributed database. The term *database* in this context is synonymous with *database management system*.

Availability and Consistency of Distributed Databases

- We might be starting to see some of the challenges to maintaining a database management system that uses multiple servers
- When two database servers must keep consistent copies of data, they incur longer times to complete a transaction
- This is acceptable in applications that require both
  - Consistency
  - High availability

  at all times
- E.g., a financial systems at a bank fall into this category
- There are applications, however, where fast database operations are more important than always maintaining consistency
- E.g., an e-commerce site might want to maintain copies of our shopping cart on two different database servers
  - If one of the servers fails, our cart is still available on the other server

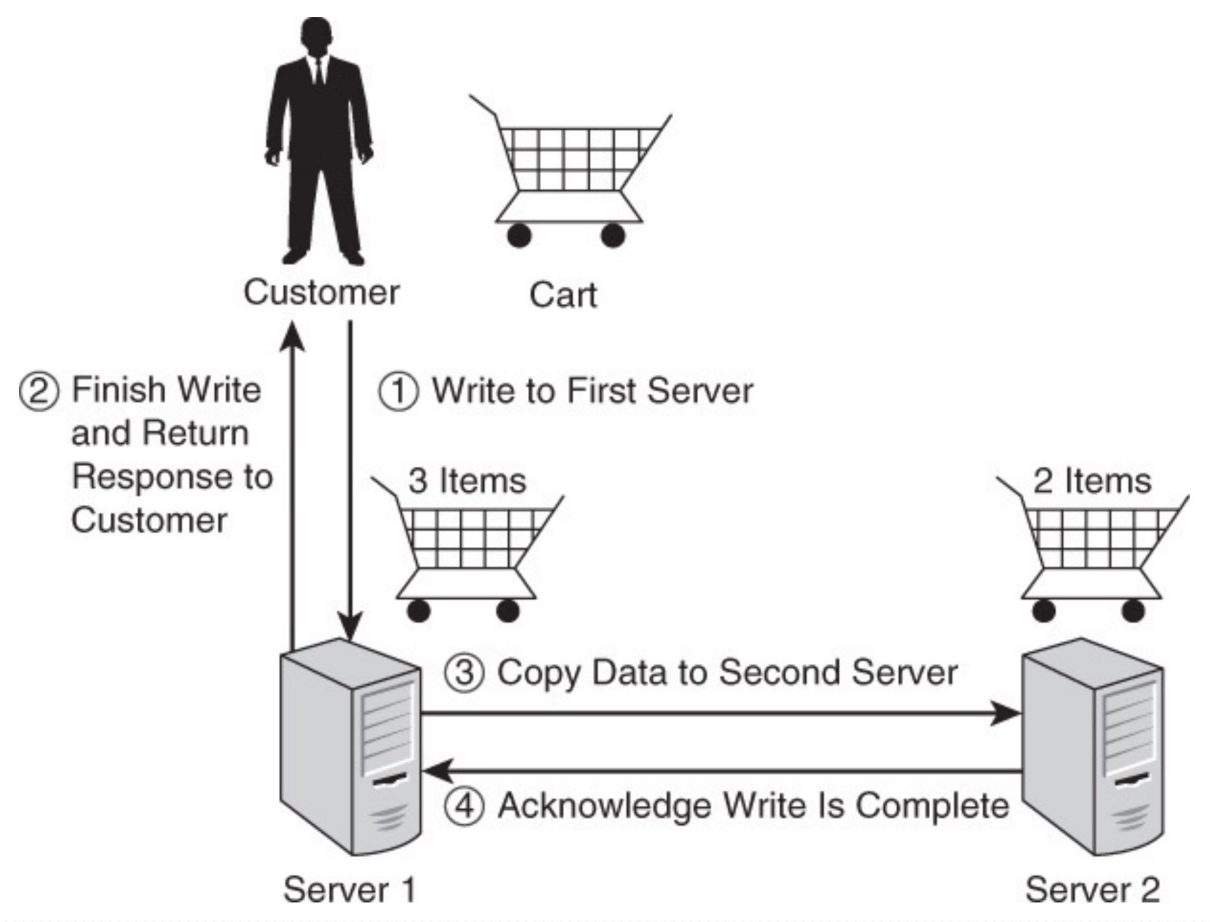
Availability and Consistency of Distributed Databases

- Imagine we are programming the user interface for an e-commerce site
- How long should the customer have to wait after clicking on an "Add to My Cart" button?
- Ideally, the interface would respond immediately so the customer could keep shopping
- If the interface feels slow and sluggish, the customer might switch to another site with faster performance
- In this case, speed is more important than always having consistent data

Availability and Consistency of Distributed Databases

- One way to deal with this problem is to write the updates to one database and then let the program know the data has been saved
- The interface can indicate to the customer that the product has been added to the cart
- While the customer receives the message that the cart has been updated, the database management system is making a copy of the newly updated data and writing it to another server
- There is a brief period when the customer's cart on the two servers is not consistent
  - The customer can continue shopping anyway
- In this case, we are willing to tolerate inconsistency for a brief period knowing that eventually the two carts will have the same products in it
- This is especially true with online shopping carts
  - There is only a small chance someone else would read that customer's cart

Availability and Consistency of Distributed Databases



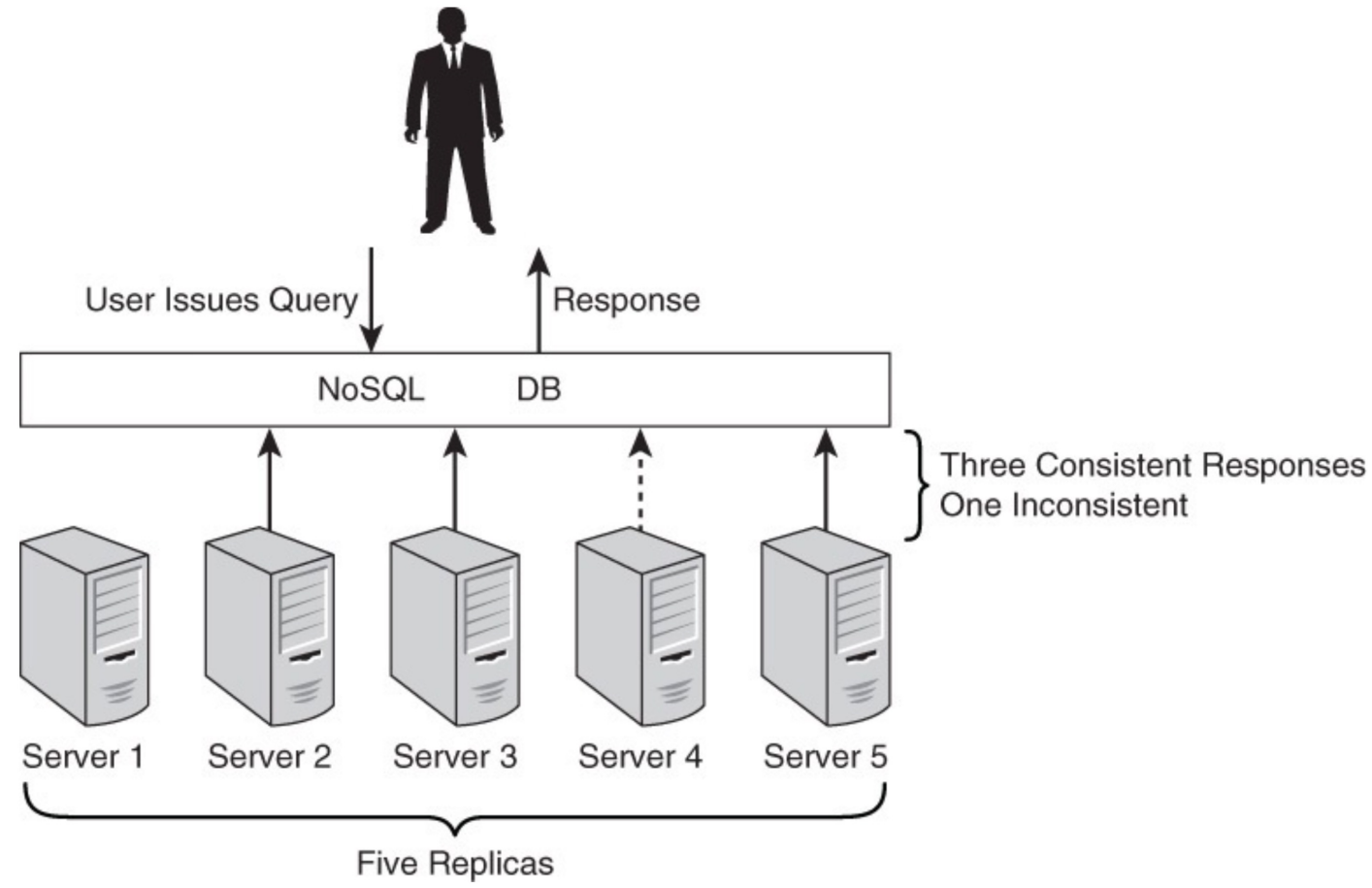
Balancing Response Times, Consistency and Durability

- NoSQL databases often implement eventual consistency
  - There might be a period where copies of data have different values
  - Eventually all copies will have the same value
- This raises the possibility of a user querying the database and getting different results from different servers within a cluster
- E.g., assume Alice has updated a customer's address in a database that implements eventual consistency
- Immediately after Alice updates the address, Bob reads that customer's address
  - Will he see the new or old address?
  - Answer is not as simple as when working with a relational database and strict consistency

Balancing Response Times, Consistency and Durability

- NoSQL databases often use the concept of quorums when working with reads and writes
- A quorum is the number of servers that must respond to a read or write operation for the operation to be considered complete
- When a read is performed, the NoSQL database reads data from, potentially, multiple servers
- Most of the time, all the servers will have consistent data
- However, while the database copies data from one of the servers to the other servers storing replicas, the replica servers may have inconsistent data
- One way to determine the correct response to any read operation is to query all servers storing that data
- The database counts the number of distinct response values and returns the one that meets or exceeds a configurable threshold

Balancing Response Times, Consistency and Durability



Balancing Response Times, Consistency and Durability

- We can vary the threshold to improve response time or consistency
- If the read threshold is set to 1, we get a fast response
- The lower the threshold, the faster the response
  - The higher the risk of returning inconsistent data
- Using the previous example, if the read threshold is set to 5, we would guarantee consistent reads
  - The query would return only after all replicas have been updated and could lead to longer response times
- We can also adjust a write threshold to balance response time and durability
  - Durability is the maintaining a correct copy of data for long periods of time
- A write operation is considered complete when a minimum number of replicas have been written to persistent storage

Balancing Response Times, Consistency and Durability

Caution

If the write threshold is set to 1, then the write is complete once a single server writes the data to persistent storage. This leads to fast response times but poor durability. If that one server or its storage system fails, the data is lost.

Balancing Response Times, Consistency and Durability

- Assume we are working the five-server cluster described previously
- If the data is replicated across three servers, we set the write threshold to 3
  - Three copies must be written to persistent storage before the write completes
- If we set the threshold to 2, our data would be written to two servers before completing the write operation
  - Other copies would be written later
- Setting the write threshold to at least 2 provides for durability
  - Setting the number of replicas higher than the threshold helps improve durability without increasing the response time of write operations

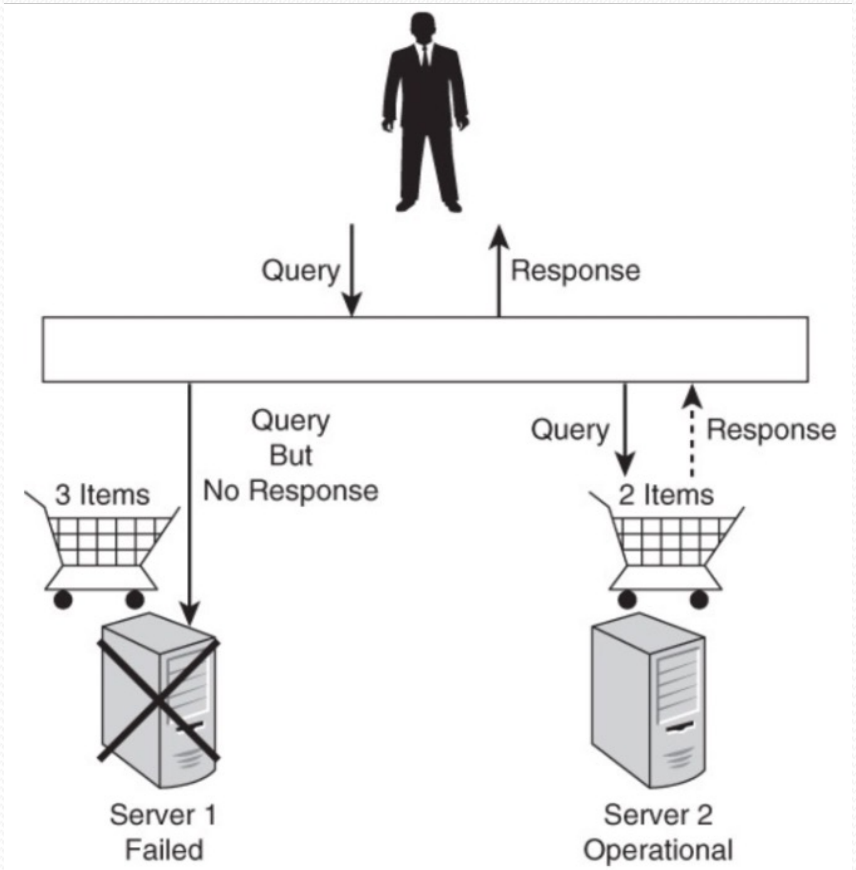
Consistency, Availability and Partitioning: CAP Theorem

- The **CAP** theorem (Brewer's …) states that distributed databases cannot have
  - **Consistency** (C)
    - consistent copies of data on different servers
  - **Availability** (A)
    - providing a response to any query
  - **Partition** protection/tolerance (P)
    - if a network that connects two or more database servers fails, the servers will still be available with consistent data

  all at the same time

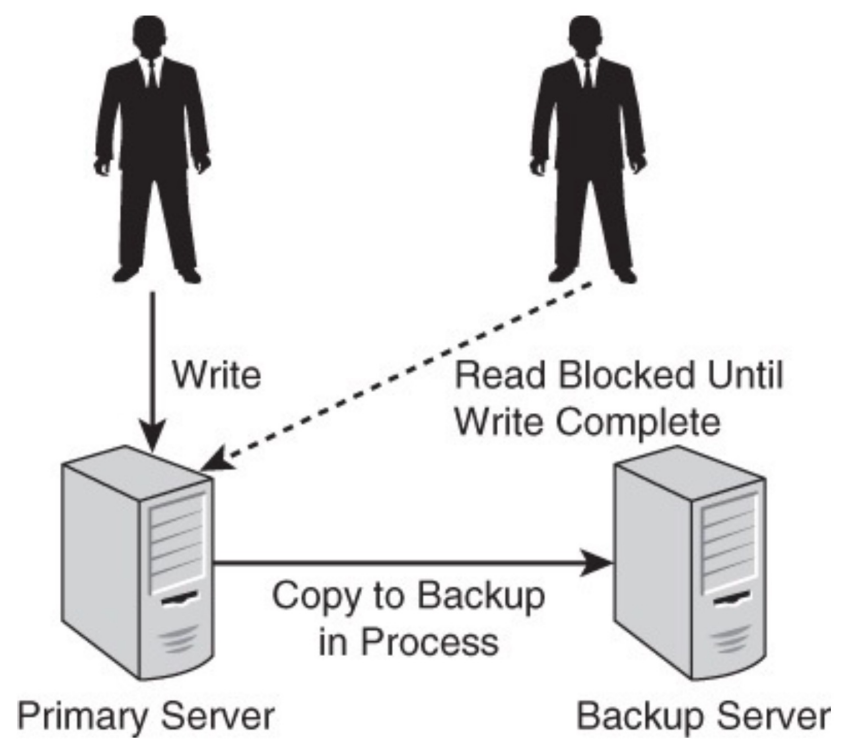
Consistency, Availability and Partitioning: CAP Theorem

- We saw in a previous example of the e-commerce shopping cart that it is possible to 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, but 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



Consistency, Availability and Partitioning: CAP Theorem

- We saw in a previous example of the two-phase commit that we can have consistency but at the risk of the most recent data not being available for a brief period
- 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



Consistency, Availability and Partitioning: CAP Theorem

- Partition protection/tolerance deals with situations in which servers cannot communicate with each other – network failure
- Partitioning is the splitting of the network into groups of devices that can communicate with each other from those that cannot
- Partitioning has multiple meanings in data management
- CAP theorem:
  - Partitioning is the inability to send messages between database servers
- If database servers running the same distributed database are partitioned by a network failure, we could
  - continue to allow both to respond to queries and preserve availability but at the risk of them becoming inconsistent
  - disable one so that only one of the servers responds to queries
    - Avoids inconsistent data to users querying different servers at the cost of availability

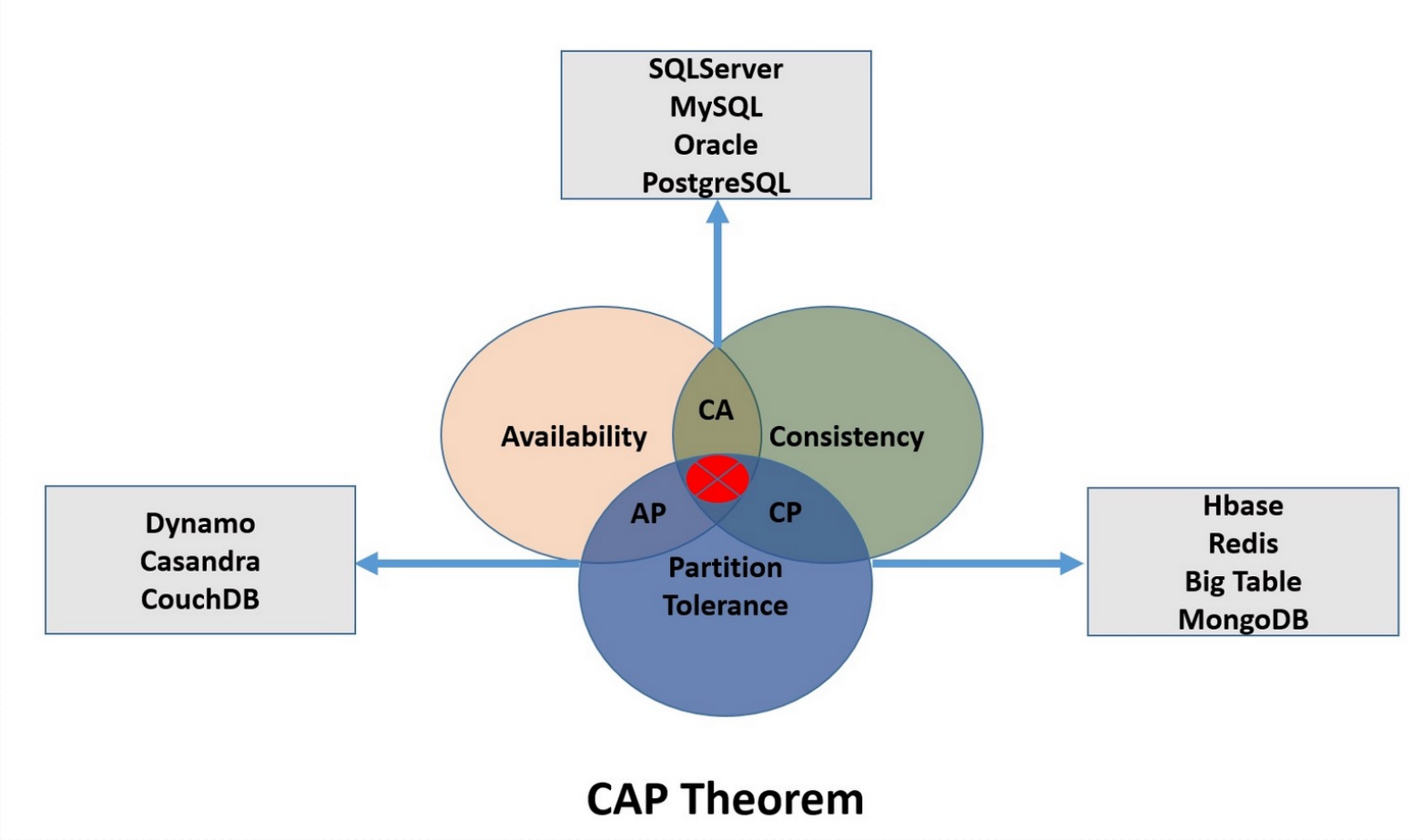
Consistency, Availability and Partitioning: CAP Theorem

- From a practical standpoint, network partitions are rare
  - At least in local area networks (LAN)
- We can imagine a wide area network (WAN) with slow network connections and low throughput that could experience network outages
  - E.g., older satellite connections to remote areas
- This means that database application designers must deal with the trade-offs between consistency and availability more than issues with partitioning

Consistency, Availability and Partitioning: CAP Theorem

- Designers of NoSQL database management systems must determine how to balance varying needs for
  - Consistency
  - Availability
  - Partitioning protection/tolerance
- NoSQL database designers can provide configuration mechanisms that allow users of the database to specify their preferred settings rather than making a single choice for all users of the database management system
- Application designers could make use of NoSQL database configuration options to make the availability-consistency trade-off decision at fine-grained levels
  - Such as different types of data in the database
- The only limitation is the configuration options provided in the NoSQL database management system used by the application

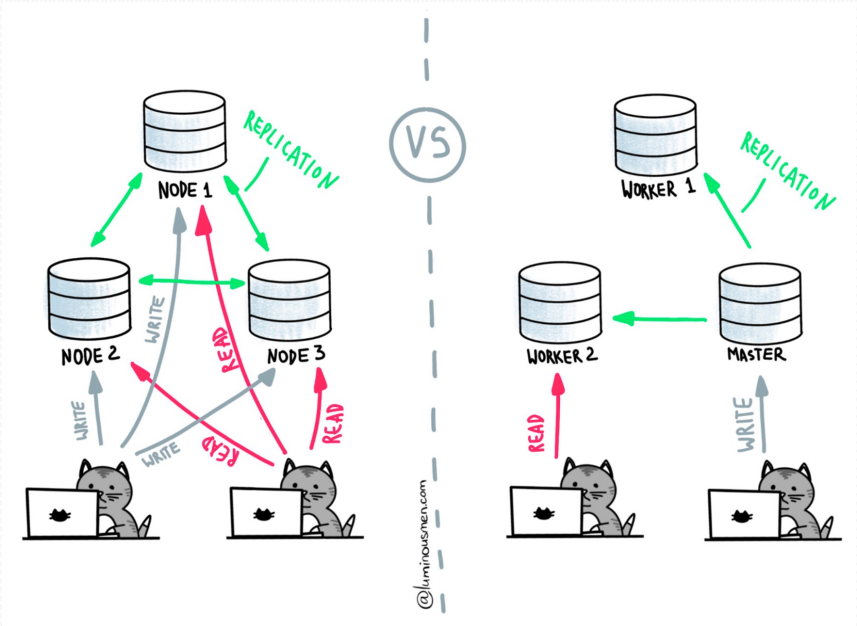
Consistency, Availability and Partitioning: CAP Theorem



ACID and BASE

- ACID is an acronym for properties common to relational database management systems
- BASE is an acronym for properties common to NoSQL databases

ACID



BASE

ACID: Atomicity, Consistency, Isolation, and Durability

- **Atomicity (A):**
- Describes a unit that cannot be further divided
- The word atom comes from the Greek atomos, which means indivisible
- In the previous discussion about transactions, we learned that all the steps had to complete or none of them completed
  - Such as transferring funds from our savings account to our checking account
- In essence, the set of steps is indivisible
- We must complete all of them as a single indivisible unit
  - Or we complete none of them

ACID: Atomicity, Consistency, Isolation, and Durability

- **Consistency (C):**
- In relational databases, this is known as strict consistency
- A transaction does not leave a database in a state that violates the integrity of data
- Transferring $100 from our savings account to our checking account must end with either
  a. $100 more in our checking account and $100 less in our savings account
  b. both accounts have the same amount as they had at the start of the transaction
- Consistency ensures no other possible state could result after a transfer operation

ACID: Atomicity, Consistency, Isolation, and Durability

- **Isolation (I):**
- Isolated transactions are not visible to other users until transactions are complete
- E.g., in the case of a bank transfer from a savings account to a checking account, someone could not read our account balances while the funds are being deducted from our savings account but before they are added to our checking account
- Databases can allow different levels of isolation
  - This can allow lost updates in which a query returns data that does not reflect the most recent update because the update operation has not finished

ACID: Atomicity, Consistency, Isolation, and Durability

- **Durability (D):**
- This means that once a transaction or operation is completed, it will remain even in the event of a power loss
  - This means that data is stored on disk, flash, or other persistent media
- Relational database management systems are designed to support ACID transactions
- NoSQL databases typically support BASE transactions
  - Some NoSQL databases also provide some level of support for ACID transactions

BASE:
Basically Available, Soft State, Eventually Consistent

- **Basically Available (BA)**:
- This means that there can be a partial failure in some parts of the distributed system and the rest of the system continues to function
- E.g., if a NoSQL database is running on 10 servers without replicating data and one of the servers fails, then 10% of the users' queries would fail, but 90% would succeed
- NoSQL databases often keep multiple copies of data on different servers
  - Allowing the database to respond to queries even if one of the servers has failed

BASE:
Basically Available, Soft State, Eventually Consistent

- **Soft State (S):**
- In computer science, the term means data will expire if it is not refreshed
- In NoSQL operations, it refers to the fact that data may eventually be overwritten with more recent data
- This property overlaps with the third property of BASE transactions, eventually consistent

BASE:
Basically Available, Soft State, Eventually Consistent

- **Eventually Consistent (E):**
- This means that there may be times when the database is in an inconsistent state
- E.g., some NoSQL databases keep multiple copies of data on multiple servers
  - There is a possibility that the multiple copies may not be consistent for a short period of time
- This occurs when a user or a program updates one copy of the data and other copies continue to have the old version of the data
- Eventually, the replication mechanism in the NoSQL database will update all copies
  - In the meantime, the copies are inconsistent

BASE:
Basically Available, Soft State, Eventually Consistent

- The time it takes to update all copies depends on several factors, such as the
  - load on the system
  - speed of the network
- Consider a database that maintains three copies of data
- A user updates their address in one server
- The NoSQL database management system automatically updates the other two copies
- One of the other copies is on a server in the same local area network
  - The update happens quickly
- The other server is in a data center thousands of miles away
  - There is a time delay in updating the third copy
- A user querying the third server, while the update is in progress, might get the user's old address while someone querying the first server gets the new address

Types of Eventual Consistency

- **Casual consistency**
  - Ensures the database reflects the order in which operations were updated
- **Read-your-writes consistency**
  - Once we have updated a record, all our reads of that record will return the updated value
- **Session consistency**
  - Ensures read-your-writes consistency during a session
- **Monotonic read consistency**
  - Ensures that if we issue a query and see a result, we will never see an earlier version of the value
- **Monotonic write consistency**
  - Ensures that if we were to issue several update commands, they would be executed in the order we issued them

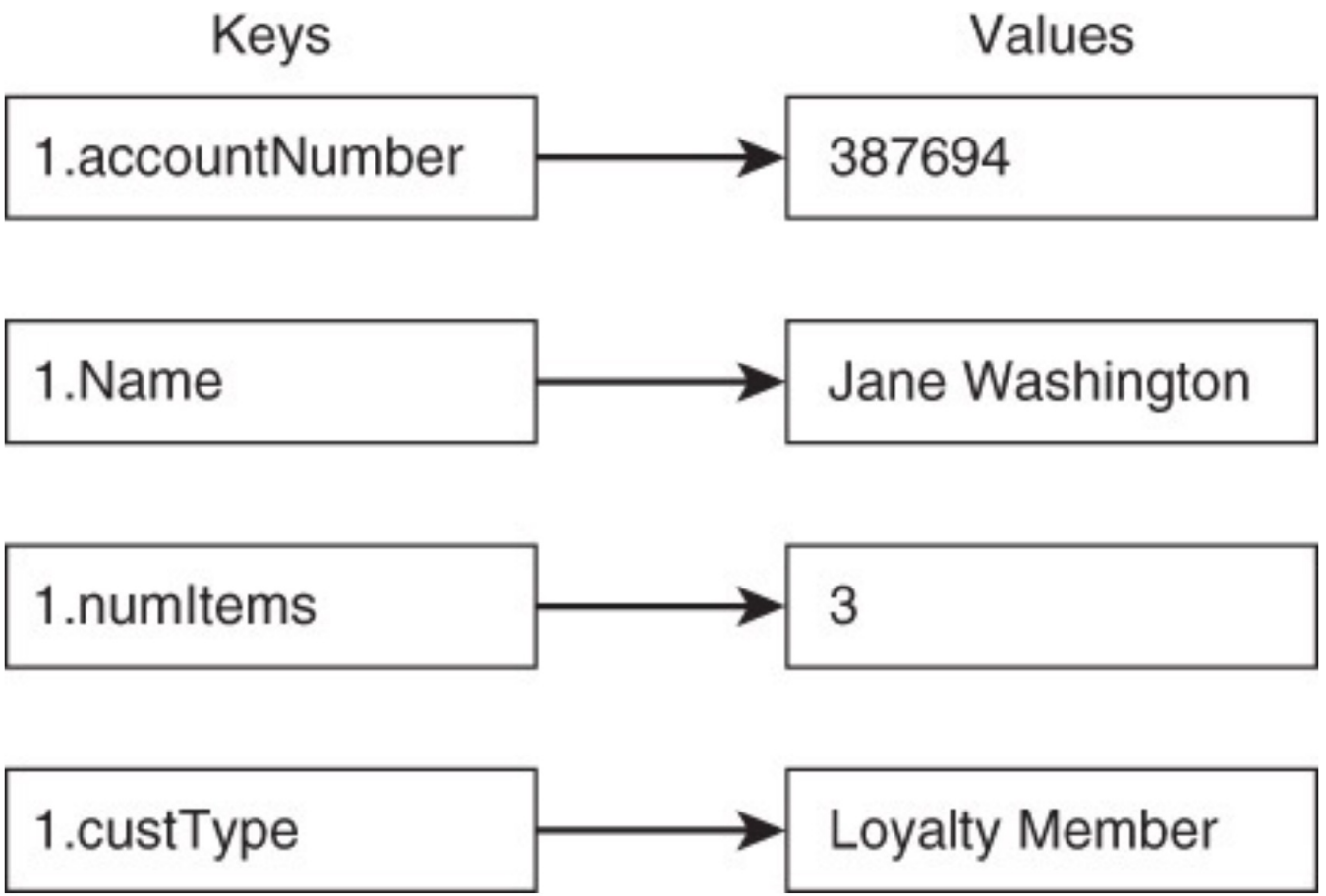
Types of NoSQL Databases

- **Key-value pair databases**
- **Document databases**
- **Column family store databases**
- **Graph databases**

Key-Value Pair Databases

- These databases are modeled on two components:
  - Keys
  - Values
- Keys are identifiers associated with values
- Let's assume we are building a key-generating program for an e-commerce website
- We realize that we need to track five things about each visitor to our site:
  - customer's account number
  - name
  - address
  - number of items in the shopping cart
  - customer type indicator
    - enrolled in the company's loyalty program

Key-Value Pair Databases



Document Databases

- Document databases (document-oriented databases) use a key-value approach to storing data but with important differences from key-value databases
- A document database stores values as documents
- Documents are semistructured entities, typically in a standard format such as
  - JavaScript Object Notation (JSON)
  - Extensible Markup Language (XML)
- When the term document is used in this context, it does **NOT** refer to
  - word processing
  - other office productivity files
- It refers to data structures that are stored as
  - strings
  - binary representations of strings

Documents

- Document databases store multiple attributes in a single document
- One of the most important characteristics of document databases is we do not have to define a fixed schema before we add data to the database
- Adding a document to the database creates the underlying data structures needed to support the document
- The lack of a fixed schema gives developers more flexibility with document databases than with relational databases
- E.g., employees can have different attributes

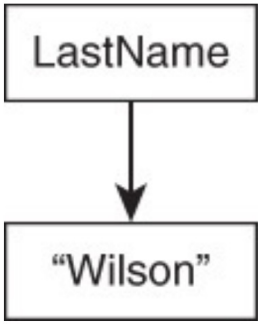
```
{
        "firstName": "Alice",
        "lastName": "Johnson",
        "position": "CFO",
        "officeNumber": "2-120",
        "officePhone": "555-222-3456"
}

{
        "firstName": "Bob",
        "lastName": "Wilson",
        "position": "Manager",
        "officeNumber": "2-130",
        "officePhone": "555-222-3478",
        "hireDate": "1-Feb-2010",
        "terminationDate": "12-Aug-2014"
}

db.employees.find( { position: "Manager" })
```

Column Family Databases

- Column family databases are perhaps the most complex of the NoSQL database types, at least in terms of the basic building block structures
- Column family databases share some terms with relational databases, such as rows and columns, but we must be careful to understand important differences between these structures
- A column is a basic unit of storage in a column family database
- A column is a name and a value

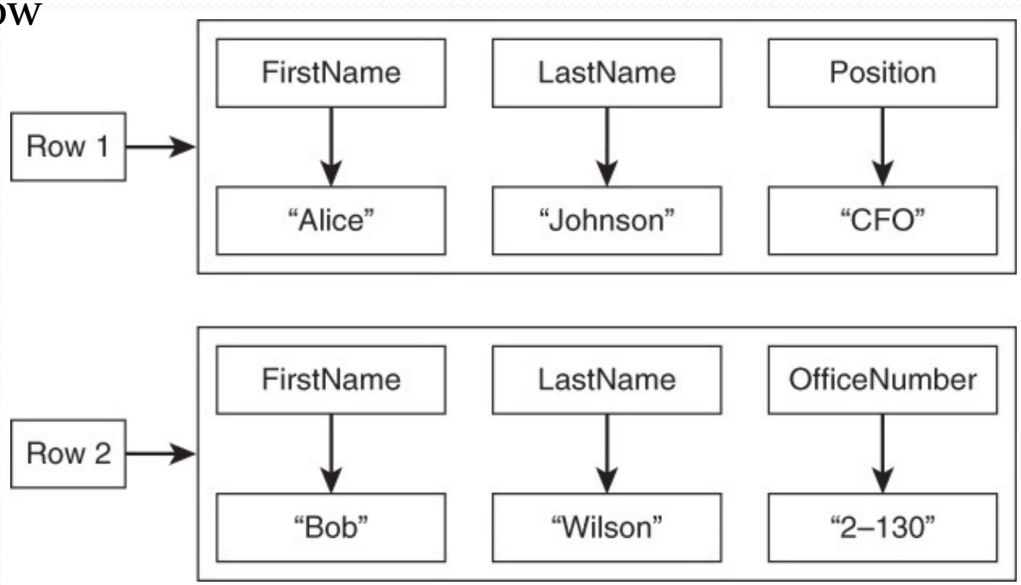


Name

Value

Column Family Databases

- A set of columns makes up a row
- Rows can have
  - the same columns
  - different columns
- As in document databases, column family databases do not require a predefined fixed schema
- Developers can add columns as needed

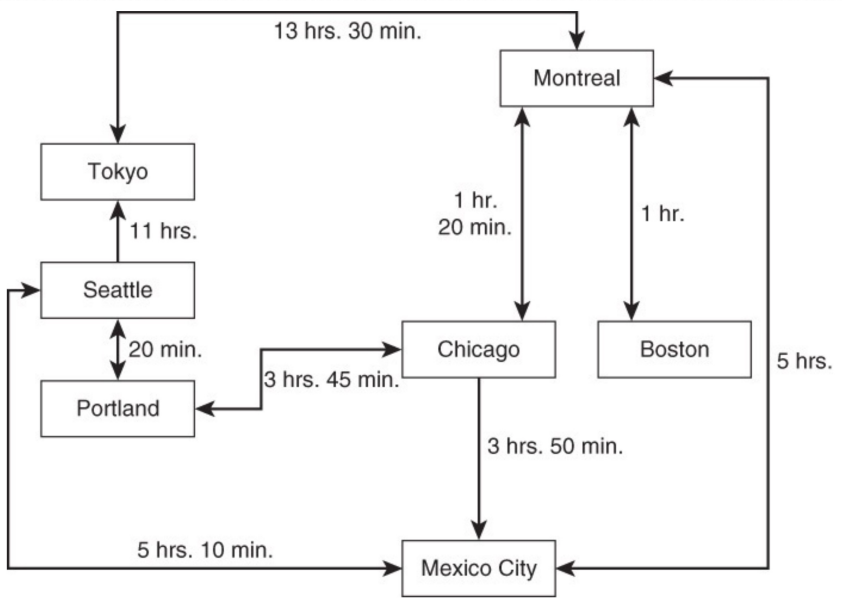


- Rows can have different sets of columns and super columns
- Column family databases are designed for rows with many columns
  - They can support millions of columns

Graph Databases

- Graph databases are the most specialized of the four NoSQL databases discussed here
- Instead of modeling data using columns and rows, a graph database uses structures called nodes and relationships
- A node (vertices) is an object that has an identifier and a set of attributes
- A relationship (edge) is a link between two nodes that contain attributes about that relation
- Nodes can represent people, and relationships can represent their friendships in social networks
- A node could be a city, and a relationship between cities could be used to store information about the distance and travel time between cities

Graph Databases



Chicago

Airports : [

{Name :
"O' Hare"
Symbol : ORD},

{Name : Midway,
Symbol : MDW}
]

Population : 2,715,000,
Area : 234 Sq. Miles

Acknowledgements

- NoSQL for Mere Mortals
  - https://www.pearson.com/us/higher-education/program/Sullivan-No-SQL-for-Mere-Mortals/PGM5054.html
- Facing Issues on IT
  - https://facingissuesonit.com/2020/02/24/cap-theorem/
- luminousmen.com
  - https://luminousmen.com/post/acid-vs-base-comparison-of-two-design-philosophies/