Distributed Systems II

Software Architecture

Brae Webb & Richard Thomas

March 31, 2025

Distributed Systems Series

Distributed I	*Reliability* and *scalability* of *stateless* systems
Distributed II	*Complexities* of *stateful* systems
Distributed III	*Hard problems* in distributed systems

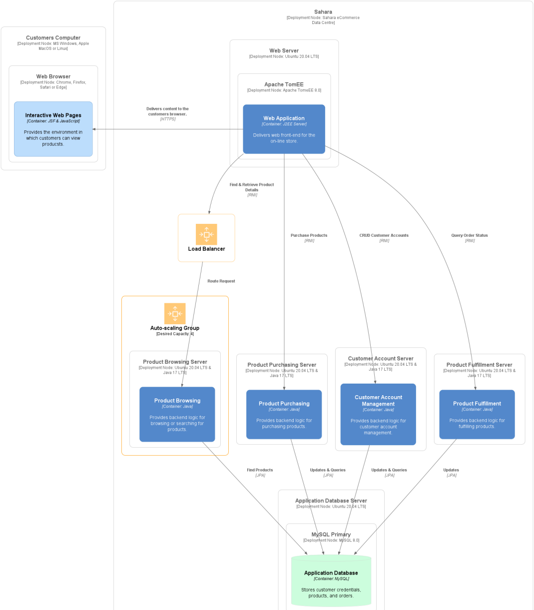
Distributed Systems Series

Distributed I	Reliability and scalability of stateless systems
Distributed II	*Complexities* of *stateful* systems
Distributed III	Hard problems in distributed systems

Previously in Distributed I: Benefits...

- Improved *reliability*
- Improved *scalability*
- Improved *latency*

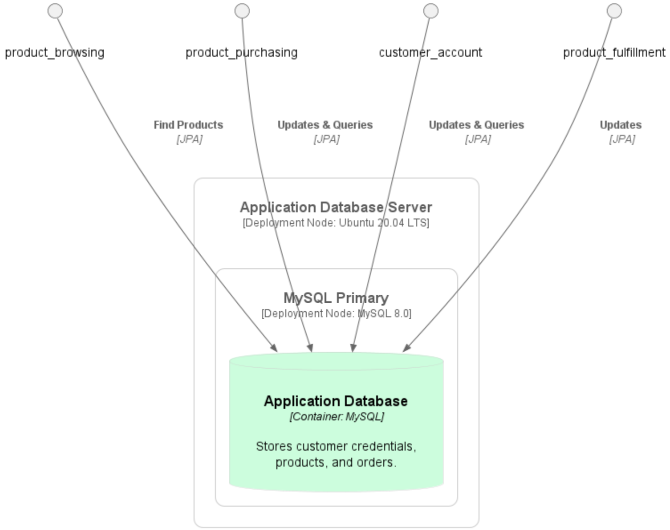
Previously in Distributed I. . .



Question

What is the *problem*?

Database



product_browsing
product_purchasing
customer_account
product_fulfillment
Find Products
[JPA]
Updates & Queries
[JPA]
Updates & Queries
[JPA]
Updates
[JPA]
Application Database Server
[Deployment Node: Ubuntu 20.04 LTS]
MySQL Primary
[Deployment Node: MySQL 8.0]
Application Database
[Container: MySQL]
Stores customer credentials, products, and orders.

Stateless vs. Stateful Systems

Stateless Does *not* utilise *persistent data*. Or: each request is independent.

Stateful Does utilise *persistent data*. Or: the server or service remembers and uses data from previous interactions.

Disclaimer

This is *not* a database course

Advanced Database Systems (INFS3200)

Course level

Undergraduate

Faculty

Engineering, Architecture & Information Technology

School

Info Tech & Elec Engineering

Units

2

Duration

One Semester

Class contact

2 Lecture hours, 1 Tutorial hour, 1 Practical or Laboratory hour

Incompatible

INFS7907

Prerequisite

INFS2200

Assessment methods

Current course offerings

Course offerings	Location	Mode	Course Profile
Semester 1, 2022	St Lucia	Internal	COURSE PROFILE
Semester 1, 2022	External	External	COURSE PROFILE
Semester 2, 2022	External	External	PROFILE UNAVAILABLE
Semester 2, 2022	St Lucia	Internal	PROFILE UNAVAILABLE

Please Note: Course profiles marked as not available may still be in development.

Course description

Distributed database design, query and transaction processing, data integration, data warehousing, data cleansing, management of spatial data, and data from large scale distributed devices.

Archived offerings

Course offerings	Location	Mode	Course Profile
Semester 1, 2021	St Lucia	Flexible Delivery	COURSE PROFILE
Semester 1, 2021	External	External	COURSE PROFILE
Semester 2, 2021	External	External	COURSE PROFILE
Semester 2, 2021	St Lucia	Internal	COURSE PROFILE
Semester 1, 2020	St Lucia	Internal	COURSE PROFILE

Question

How do we fix database scaling issues?

Question

How do we fix database scaling issues?

Answer

- Replication

Question

How do we fix database scaling issues?

Answer

- Replication
- Partitioning

Question

How do we fix database scaling issues?

Answer

- Replication
- Partitioning
- Independent databases

Question

How do we fix database scaling issues?

Answer

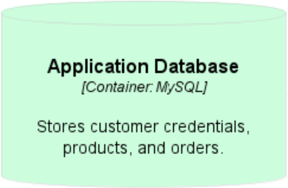
- *Replication*
- Partitioning
- Independent databases

Question

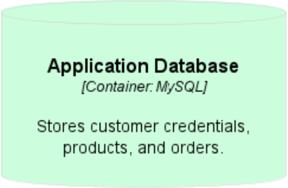
What is *replication*?

Definition 0. Replication

Data copied across multiple different machines.



product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	10	$10.00
4321	Lifelike Elephant Inflatable	5	$50.00



product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	10	$10.00
4321	Lifelike Elephant Inflatable	5	$50.00

Definition 0. Replica

Database node which stores a copy of the data.

Question

What are the advantages of *replication*?

Question

What are the advantages of *replication*?

Answer

- *Scale* our database to cope with higher loads.

Question

What are the advantages of *replication*?

Answer

- *Scale* our database to cope with higher loads.
- Provide *fault tolerance* from a single instance failure.

Question

What are the advantages of *replication*?

Answer

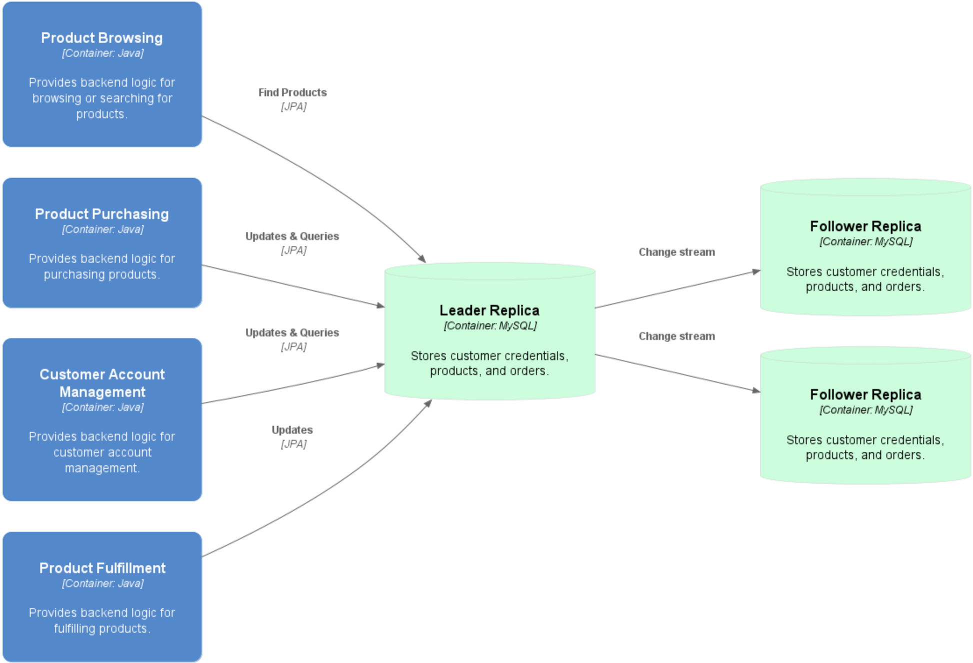
- *Scale* our database to cope with higher loads.
- Provide *fault tolerance* from a single instance failure.
- Locate instances *closer to end-users*.

Question

How do we replicate our data?

First Approach

Leader-Follower Replication


Product Browsing
[Container: Java]
Provides backend logic for browsing or searching for products.
Product Purchasing
[Container: Java]
Provides backend logic for purchasing products.
Customer Account Management
[Container: Java]
Provides backend logic for customer account management.
Product Fulfillment
[Container: Java]
Provides backend logic for fulfilling products.
Find Products
[JPA]
Updates & Queries
[JPA]
Updates & Queries
[JPA]
Updates
[JPA]
Leader Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
Change stream
Change stream
Follower Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
Follower Replica
[Container: MySQL]
Stores customer credentials, products, and orders.

Definition 0. Leader-based Replication

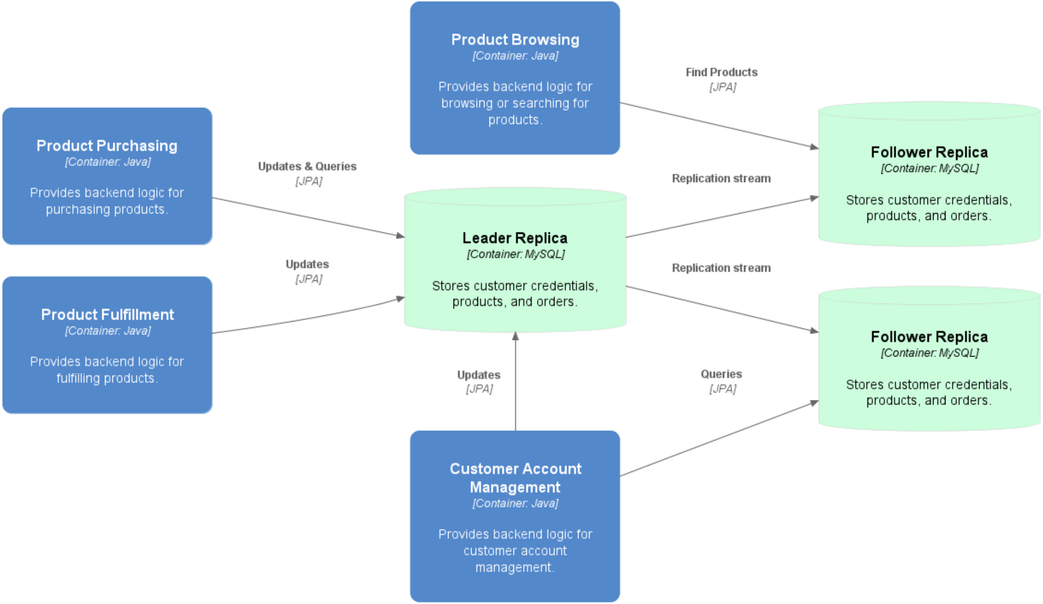
one node (the leader) handles all write operations, and multiple other nodes (followers) replicate the data and handle read operations.

Leader-based Replication

On write Writes sent to *leader*, change is propagated via change stream.

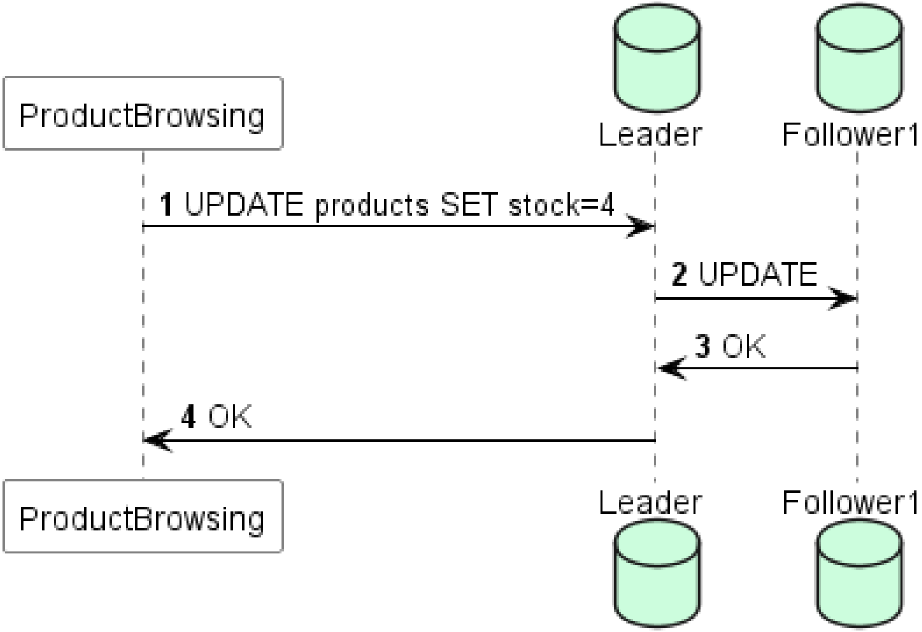
Leader-based Replication

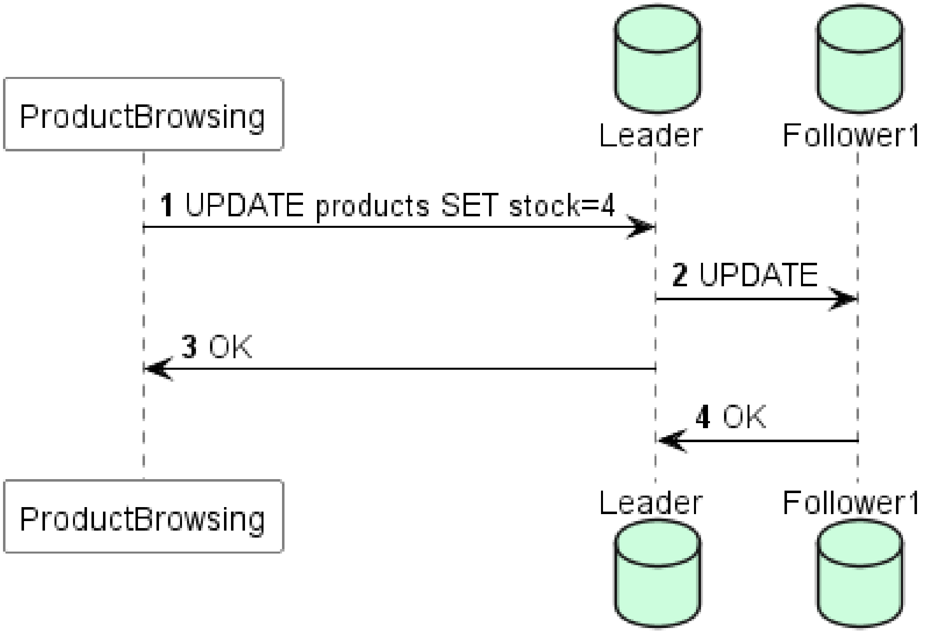
On write	Writes sent to *leader*, change is propagated via change stream.
On read	Any *replica* can be queried.

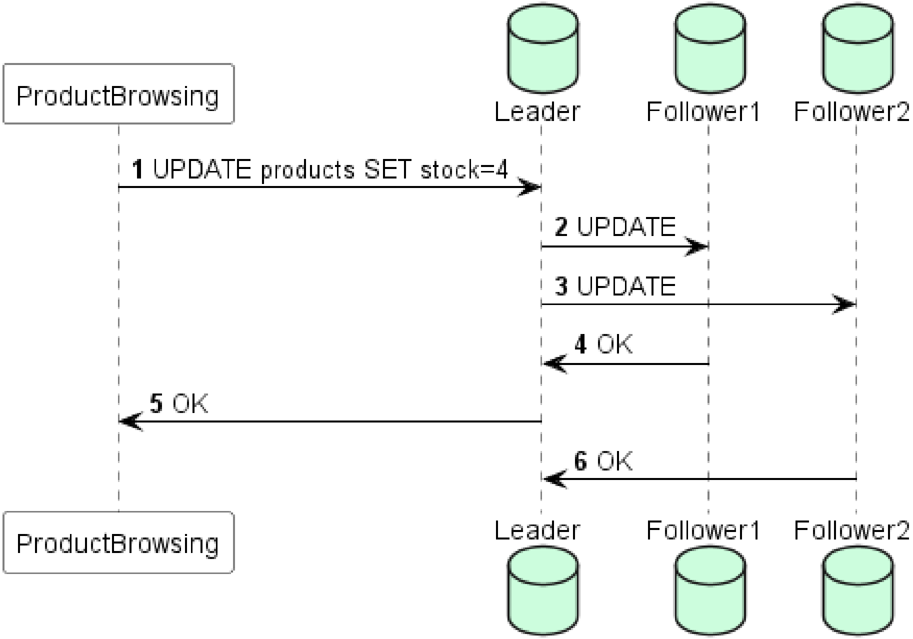

Product Browsing
[Container: Java]
Provides backend logic for browsing or searching for products.
Find Products
[JPA]
Product Purchasing
[Container: Java]
Provides backend logic for purchasing products.
Updates & Queries
[JPA]
Replication stream
Follower Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
Leader Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
Updates
[JPA]
Replication stream
Product Fulfillment
[Container: Java]
Provides backend logic for fulfilling products.
Follower Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
Updates
[JPA]
Queries
[JPA]
Customer Account Management
[Container: Java]
Provides backend logic for customer account management.

Propagating Changes

Synchronous vs. *Asynchronous*


ProductBrowsing
Leader
Follower1
1 UPDATE products SET stock=4
2 UPDATE
3 OK
4 OK
ProductBrowsing
Leader
Follower1


ProductBrowsing
Leader
Follower1
1 UPDATE products SET stock=4
2 UPDATE
3 OK
4 OK
ProductBrowsing
Leader
Follower1



ProductBrowsing
Leader
Follower1
Follower2
1 UPDATE products SET stock=4
2 UPDATE
3 UPDATE
4 OK
5 OK
6 OK
ProductBrowsing
Leader
Follower1
Follower2

Synchronous Propagation

- Writes must propagate to *all followers* before being successful.

Synchronous Propagation

- Writes must propagate to *all followers* before being successful.
- *Any* replica goes down, *all* replicas are un-writeable.

Synchronous Propagation

- Writes must propagate to *all followers* before being successful.
- *Any* replica goes down, *all* replicas are un-writeable.
- Writes must *wait* for propagation to *all* replicas.

Asynchronous Propagation

- Writes *don't* have to *wait* for propagation.

Asynchronous Propagation

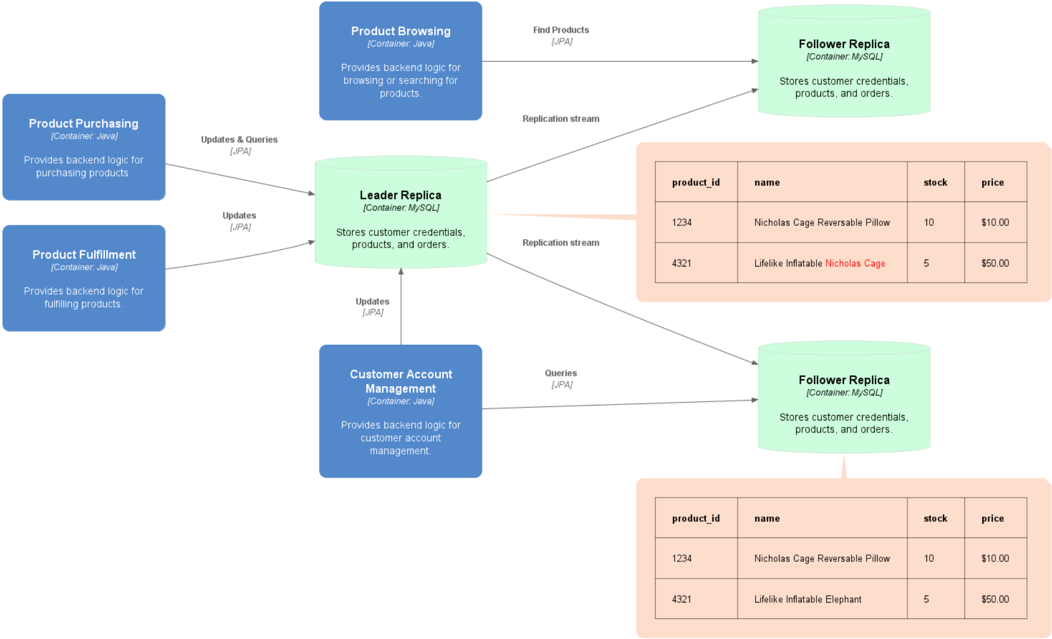
- Writes *don't* have to *wait* for propagation.
- If the leader goes down before propagating, the *write is lost*.

Asynchronous Propagation

- Writes *don't* have to *wait* for propagation.
- If the leader goes down before propagating, the *write is lost*.
- Replicas can have out-dated or *stale* data.

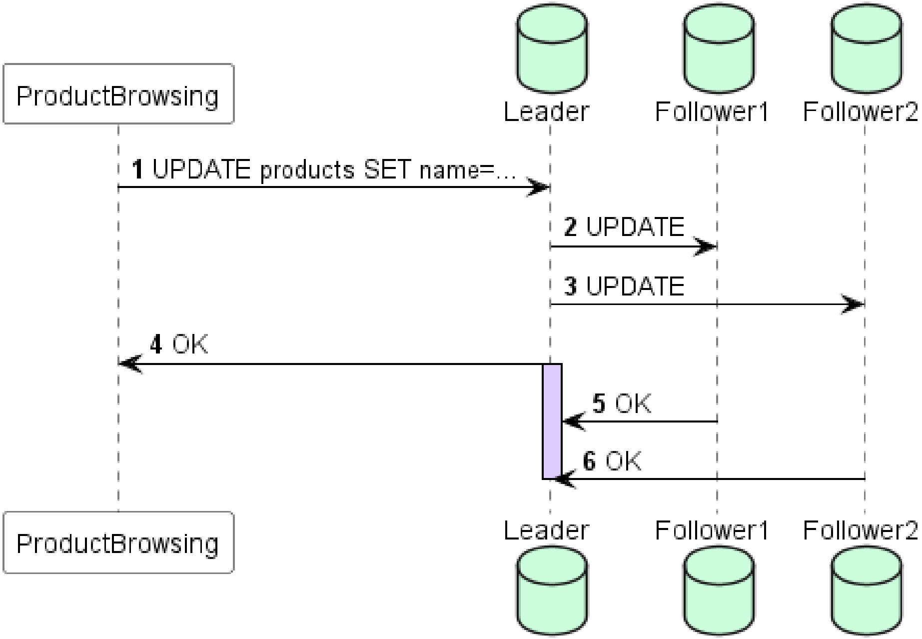
Definition 0. Stale data

Outdated or inconsistent data that does not reflect the latest updates, mainly due to replication delays, caching, or network issues in distributed systems.


Product Browsing
[Container: Java]
Provides backend logic for browsing or searching for products.
Find Products
[JPA]
Follower Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
Product Purchasing
[Container: Java]
Provides backend logic for purchasing products.
Updates & Queries
[JPA]
Replication stream
Leader Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
product_id | name | stock | price
1234 | Nicholas Cage Reversable Pillow | 10 | $10.00
4321 | Lifelike Inflatable Nicholas Cage | 5 | $50.00
Updates
[JPA]
Product Fulfillment
[Container: Java]
Provides backend logic for fulfilling products.
Replication stream
Updates
[JPA]
Customer Account Management
[Container: Java]
Provides backend logic for customer account management.
Queries
[JPA]
Follower Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
product_id | name | stock | price
1234 | Nicholas Cage Reversable Pillow | 10 | $10.00
4321 | Lifelike Inflatable Elephant | 5 | $50.00

Definition 0. Replication Lag

The time taken for replicas to update *stale* data.


ProductBrowsing
Leader
Follower1
Follower2
1 UPDATE products SET name=...
2 UPDATE
3 UPDATE
4 OK
5 OK
6 OK
ProductBrowsing
Leader
Follower1
Follower2

Eventually, all replicas must become consistent

The system is *eventually consistent*.

Eventual Consistency

Sufficient? Problems?

Brae Webb

@braewebb

Brae Webb

@braewebb

Name: Brae

Cancel Save

Brae Webb

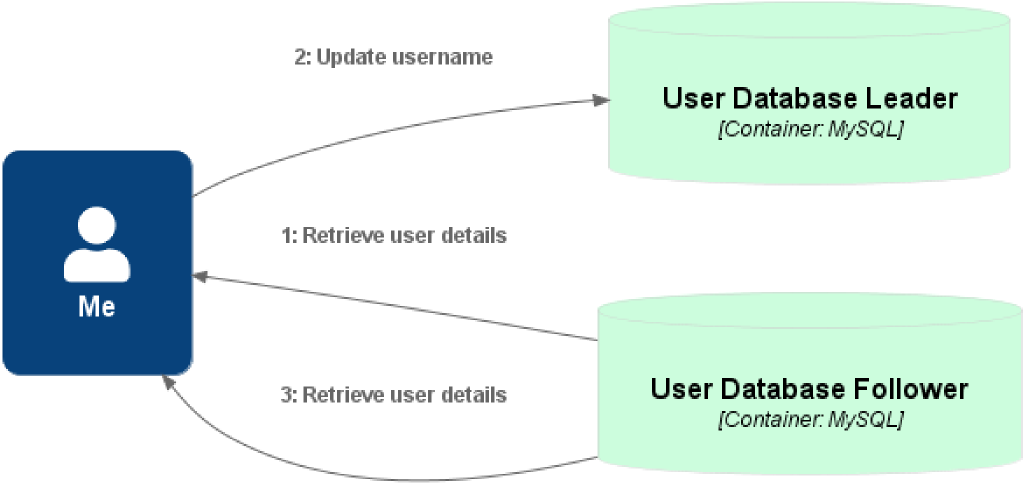
@braewebb

Name: Brae

Cancel Save

Brae Webb

@braewebb


2: Update username
User Database Leader
[Container: MySQL]
Me
1: Retrieve user details
User Database Follower
[Container: MySQL]
3: Retrieve user details

Definition 0. Read-your-writes Consistency

Users always see the updates that *they have made* (even though others see stale data).

Brae Webb

@braewebb

My fist post

Brae Webb

@braewebb

My fist post

Brae Webb

@braewebb

My first post

Brae Webb

@braewebb

My fist post

Brae Webb

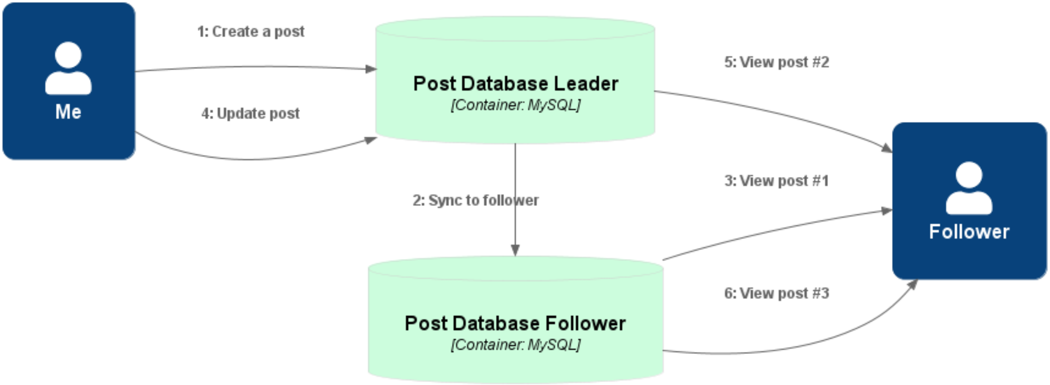
@braewebb

My first post

Brae Webb

@braewebb

My fist post


Me
1: Create a post
4: Update post
Post Database Leader
[Container: MySQL]
2: Sync to follower
Post Database Follower
[Container: MySQL]
5: View post #2
3: View post #1
6: View post #3
Follower

Definition 0. Monotonic Reads

Once a user reads an updated value, they don't later see the old value.

Definition 0. Causal Consistency

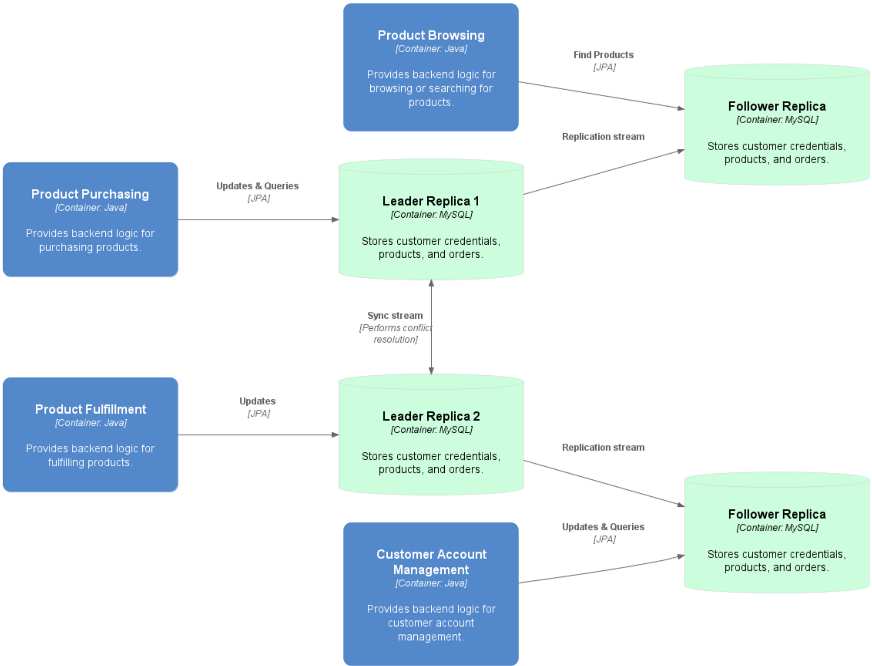
Causally related updates appear in order (e.g., comments appear under the right post).

Summary

- Leader-follower databases allow *reads to scale* more effectively.
- Asynchronous propagation weakens consistency to *eventually consistent*.
- Leader-follower databases still have a *leader write bottle-neck*.

Second approach

Multi-leader Replication


Product Browsing
[Container: Java]
Provides backend logic for browsing or searching for products.
Find Products
[JPA]
Follower Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
Replication stream
Product Purchasing
[Container: Java]
Provides backend logic for purchasing products.
Updates & Queries
[JPA]
Leader Replica 1
[Container: MySQL]
Stores customer credentials, products, and orders.
Sync stream
[Performs conflict resolution]
Product Fulfillment
[Container: Java]
Provides backend logic for fulfilling products.
Updates
[JPA]
Leader Replica 2
[Container: MySQL]
Stores customer credentials, products, and orders.
Replication stream
Follower Replica
[Container: MySQL]
Stores customer credentials, products, and orders.
Updates & Queries
[JPA]
Customer Account Management
[Container: Java]
Provides backend logic for customer account management.

Why multi-leader?

- If you have multiple leaders, you can write to any, allowing *writes to scale*.

Why multi-leader?

- If you have multiple leaders, you can write to any, allowing *writes to scale*.
- A leader going down doesn't prevent writes, giving *better fault-tolerance*.

Question

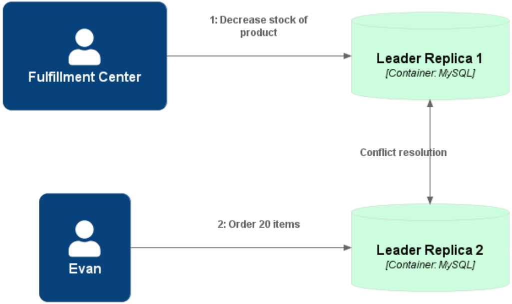
What might go wrong?

Question

What might go wrong?

Answer

Write conflicts

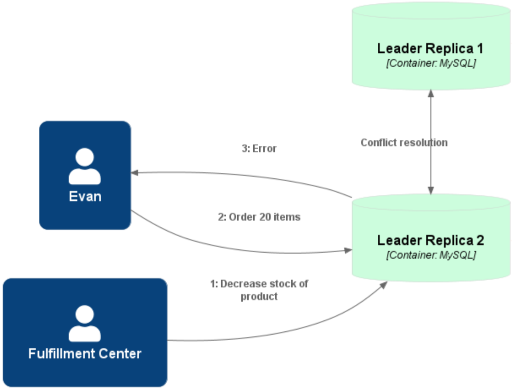


product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	0	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

Where possible

Avoid write conflicts



product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	20	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

Where impossible

Convergence

Convergence Strategies

- Assign each *write* a unique ID

Convergence Strategies

- Assign each *write* a unique ID
- Assign each *leader replica* a unique ID

Convergence Strategies

- Assign each *write* a unique ID
- Assign each *leader replica* a unique ID
- Custom resolution logic

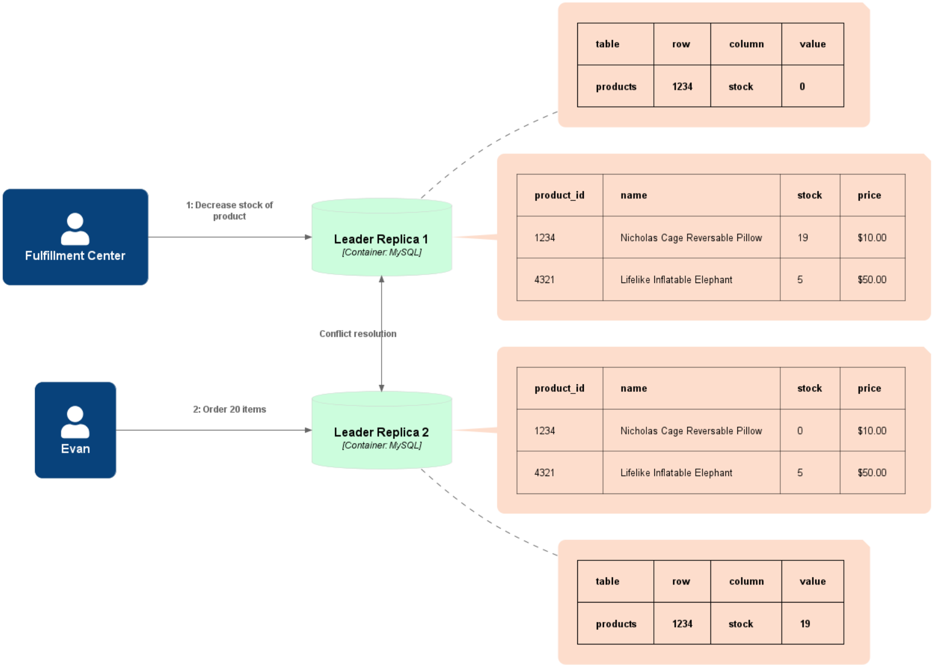


table	row	column	value
products	1234	stock	0

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	0	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

table	row	column	value
products	1234	stock	19

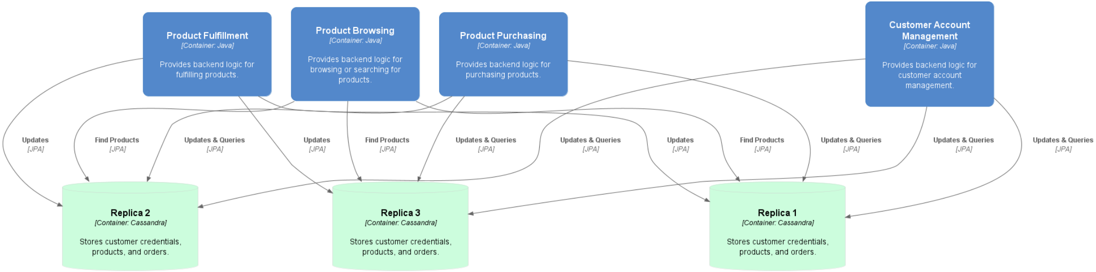
Resolving Conflicts

On Write When a conflict is first noticed, take proactive resolution action.
Example: Last Write Wins (LWW) in DynamoDB.

On Read Stores all conflicting versions on write. When a conflict is next read, ask for a resolution.
Example: *git commit* -¿ *git pull* in

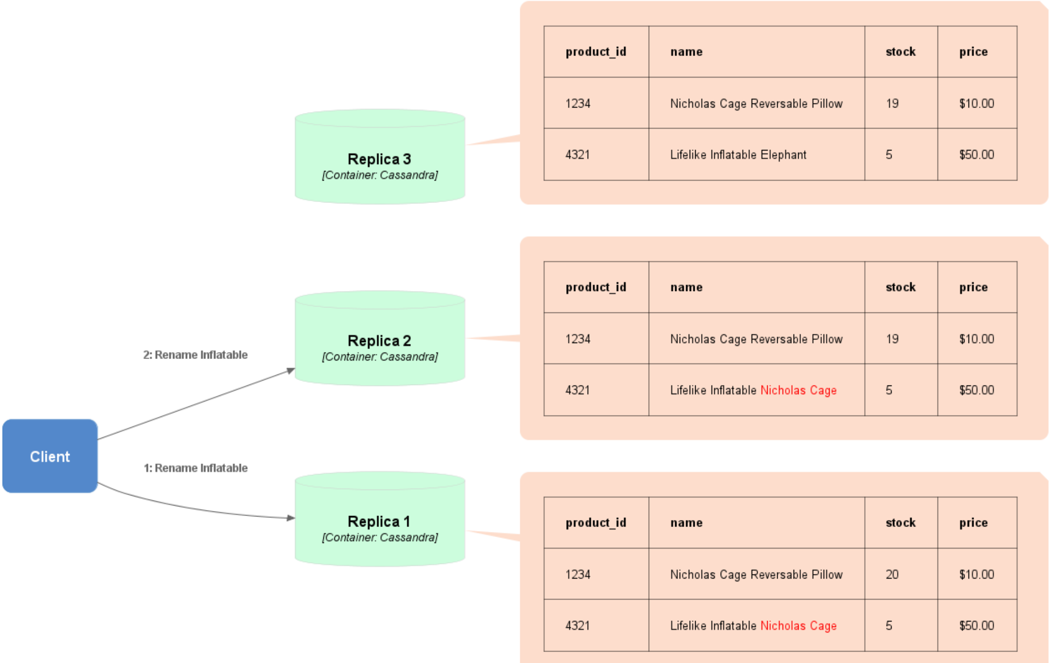
Third Approach

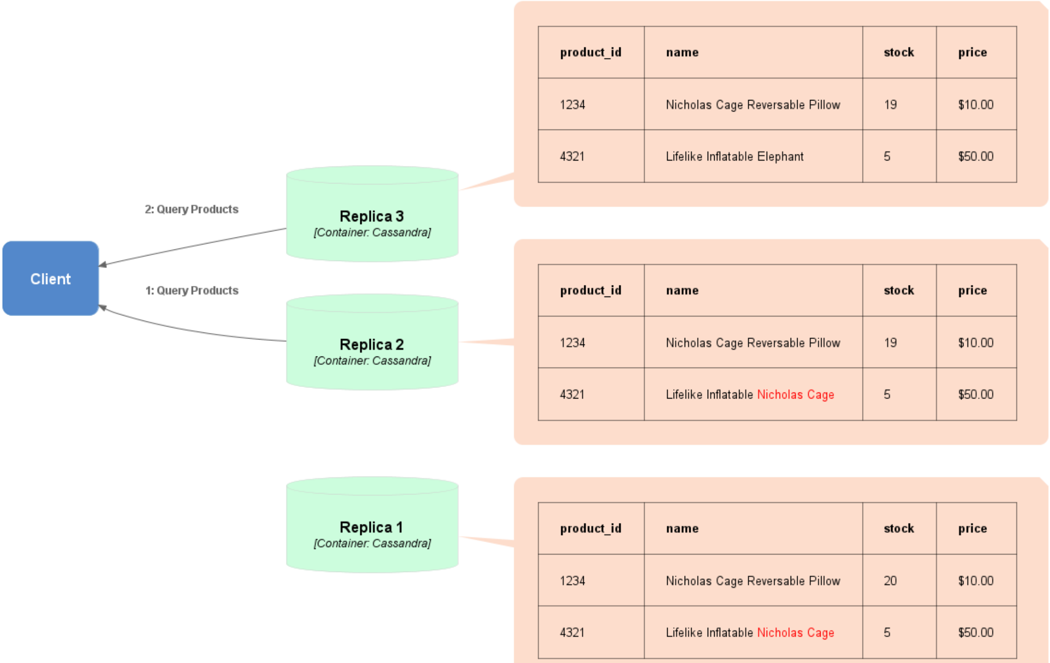
Leaderless Replication


Product Fulfillment
[Container: Java]
Provides backend logic for fulfilling products.
Product Browsing
[Container: Java]
Provides backend logic for browsing or searching for products.
Product Purchasing
[Container: Java]
Provides backend logic for purchasing products.
Customer Account Management
[Container: Java]
Provides backend logic for customer account management.
Updates
[JPA]
Find Products
[JPA]
Updates & Queries
[JPA]
Updates
[JPA]
Find Products
[JPA]
Updates & Queries
[JPA]
Updates & Queries
[JPA]
Updates
[JPA]
Find Products
[JPA]
Updates & Queries
[JPA]
Updates & Queries
[JPA]
Updates & Queries
[JPA]
Replica 2
[Container: Cassandra]
Stores customer credentials, products, and orders.
Replica 3
[Container: Cassandra]
Stores customer credentials, products, and orders.
Replica 1
[Container: Cassandra]
Stores customer credentials, products, and orders.

How do they work?

Each read/write is sent to *multiple* replicas.


product_id
name
stock
price
1234
Nicholas Cage Reversable Pillow
19
$10.00
4321
Lifelike Inflatable Elephant
5
$50.00
Replica 3
[Container: Cassandra]
product_id
name
stock
price
1234
Nicholas Cage Reversable Pillow
19
$10.00
4321
Lifelike Inflatable Nicholas Cage
5
$50.00
Replica 2
[Container: Cassandra]
2: Rename Inflatable
Client
1: Rename Inflatable
Replica 1
[Container: Cassandra]
product_id
name
stock
price
1234
Nicholas Cage Reversable Pillow
20
$10.00
4321
Lifelike Inflatable Nicholas Cage
5
$50.00



Client
2: Query Products
1: Query Products
Replica 3
[Container: Cassandra]
Replica 2
[Container: Cassandra]
Replica 1
[Container: Cassandra]

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Nicholas Cage	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	20	$10.00
4321	Lifelike Inflatable Nicholas Cage	5	$50.00

How are changes propagated?

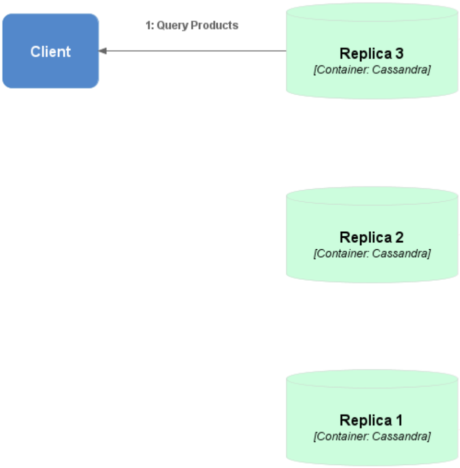
- Read Repair

How are changes propagated?

- Read Repair
- Anti-Entropy Process

Question

How do we know it's consistent?



product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Nicholas Cage	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	20	$10.00
4321	Lifelike Inflatable Nicholas Cage	5	$50.00

Question

How do we know it's consistent?

Question

How do we know it's consistent?

Answer

Quorum Reads and Writes

Quorum Consistency

$$w + r > n$$

n total replicas

w amount of replicas to *write* to

r amount of replicas to *read* from

Quorum Consistency

$$2 + 2 > 3$$

n total replicas

w amount of replicas to *write* to

r amount of replicas to *read* from

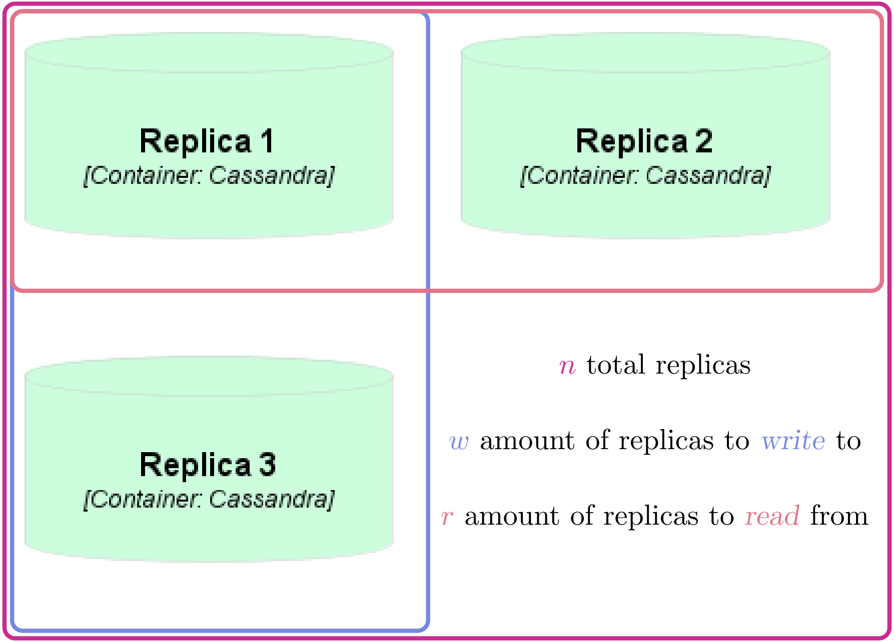
Quorum Consistency

$$1 + 3 > 3$$

n total replicas

w amount of replicas to *write* to

r amount of replicas to *read* from


Replica 1
[Container: Cassandra]
Replica 2
[Container: Cassandra]
Replica 3
[Container: Cassandra]
n total replicas
w amount of replicas to write to
r amount of replicas to read from

Question

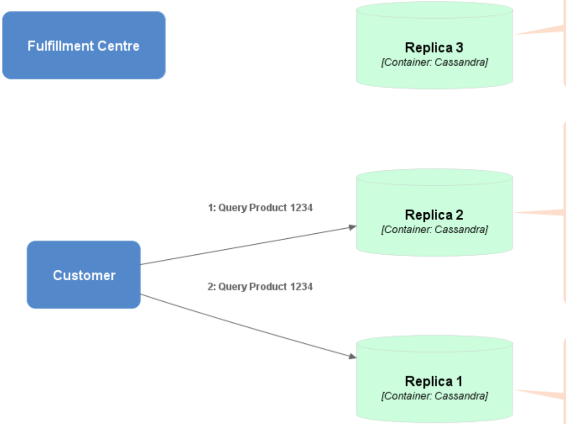
What about write conflicts?

Question

What about write conflicts?

Answer

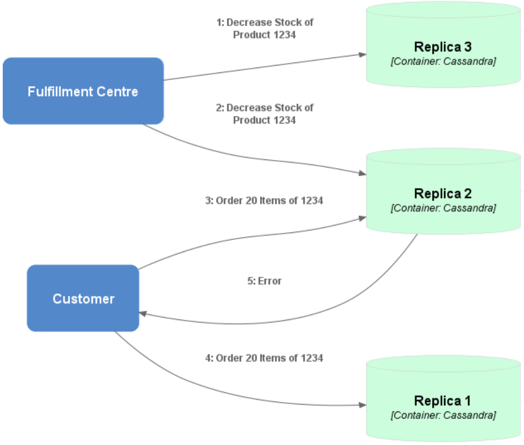
Same problem as with Multi-leader replication.



product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	20	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	20	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	20	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00



product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	19	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	20	$10.00
4321	Lifelike Inflatable Elephant	5	$50.00

Summary

- *Replication* copies data to multiple replicas.

Summary

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simpliest.

Summary

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simpliest.
- Replication introduces *eventual consistency*.

Summary

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simpliest.
- Replication introduces *eventual consistency*.
- *Multi-leader* replication scales writes as well as reads but introduces *write conflicts*.

Summary

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simpliest.
- Replication introduces *eventual consistency*.
- *Multi-leader* replication scales writes as well as reads but introduces *write conflicts*.
- *Leaderless* replication is another approach which keeps the problems of multi-leader.

Question

How do we fix database scaling issues?

Question

How do we fix database scaling issues?

Answer

- *Replication*
- Partitioning
- Independent databases

Question

How do we fix database scaling issues?

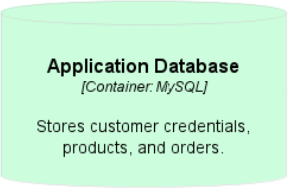
Answer

- Replication
- *Partitioning*
- Independent databases

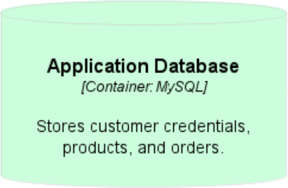
Definition 0. Partitioning

Split the data of a system onto multiple nodes.

These nodes are *partitions*.



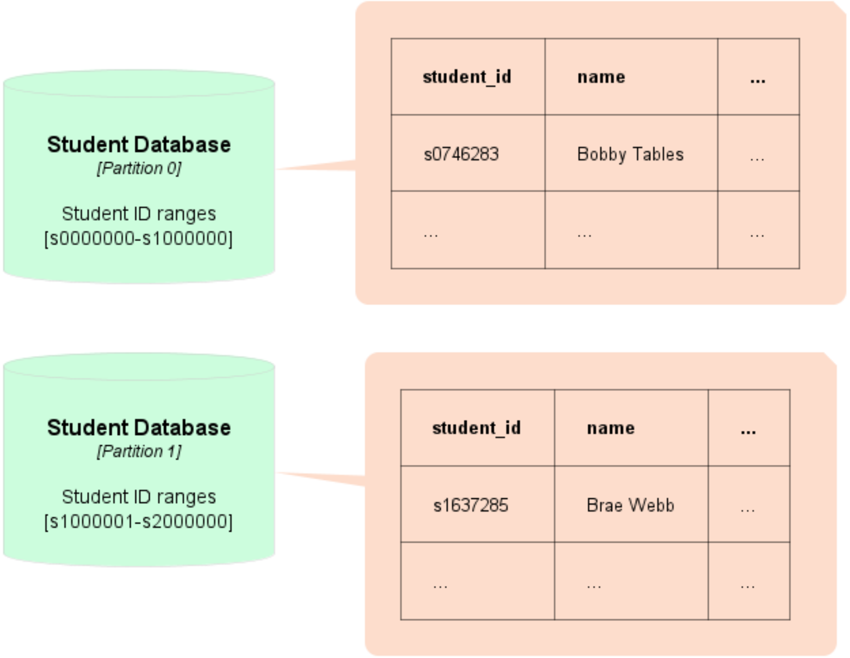
product_id	name	stock	price
4321	Lifelike Elephant Inflatable	5	$50.00



product_id	name	stock	price
1234	Nicholas Cage Reversable Pillow	10	$10.00

Question

How should we decide which data is stored where?


Student Database
[Partition 0]
Student ID ranges
[s0000000-s1000000]
student_id
name
...
s0746283
Bobby Tables
...
...
...
...
Student Database
[Partition 1]
Student ID ranges
[s1000001-s2000000]
student_id
name
...
s1637285
Brae Webb
...
...
...
...

Question

What is the problem with this?

Question

What is the problem with this?

Answer

Over time some partitions become inactive, while others receive almost all load.

Question

How should we decide where data is stored?

Question

How should we decide where data is stored?

Answer

Maximize spread of requests, avoiding *skewing*.

Question

Have we seen this before?

Question

Have we seen this before?

Answer

Hashing?

Question

What is the problem with this?

Question

What is the problem with this?

Answer

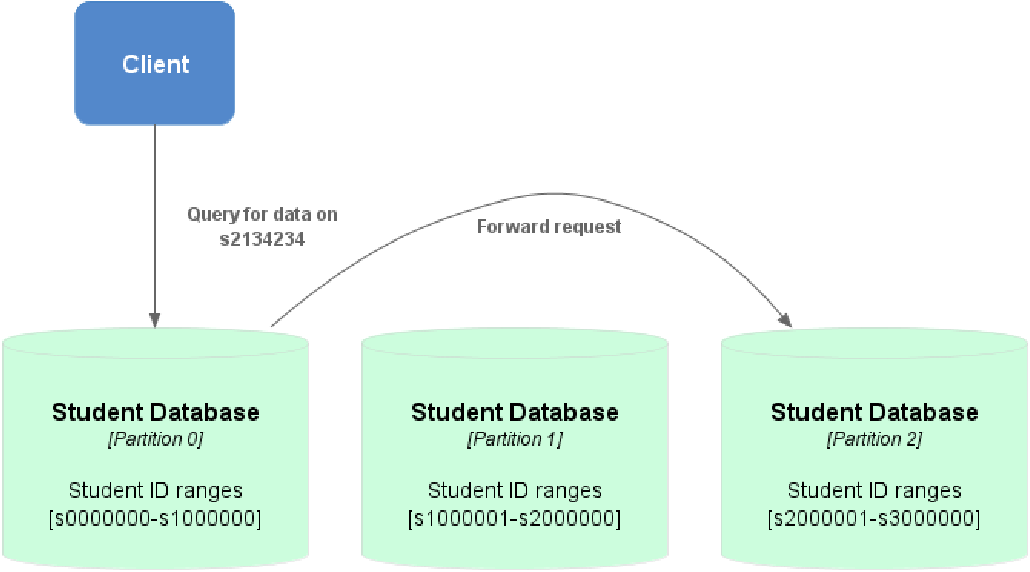
Range queries are inefficient, i.e. get all students between s4444444 and s4565656.

Question

How do we route queries?

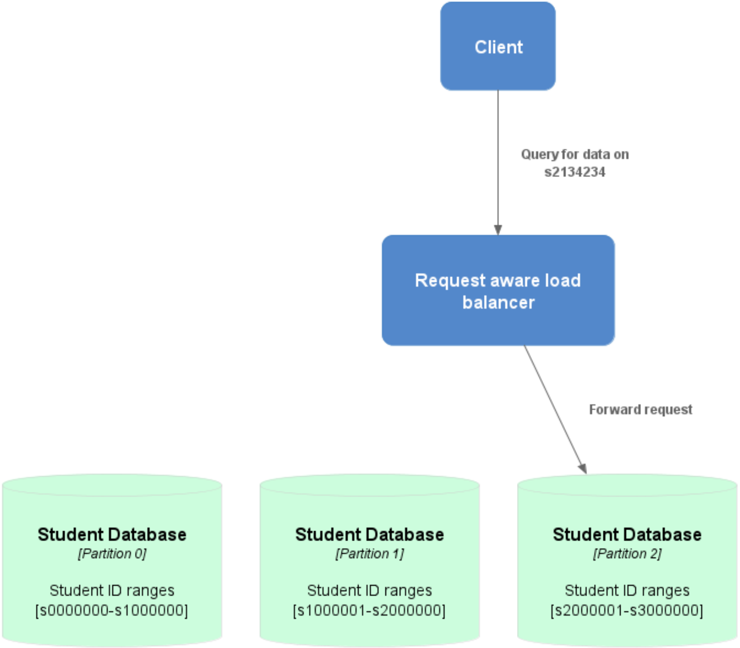
Query-Insensitive Load Balancer

Randomly route to any node, responsibility of the node to re-route to the correct node.


Client
Query for data on s2134234
Forward request
Student Database
[Partition 0]
Student ID ranges [s0000000-s1000000]
Student Database
[Partition 1]
Student ID ranges [s1000001-s2000000]
Student Database
[Partition 2]
Student ID ranges [s2000001-s3000000]

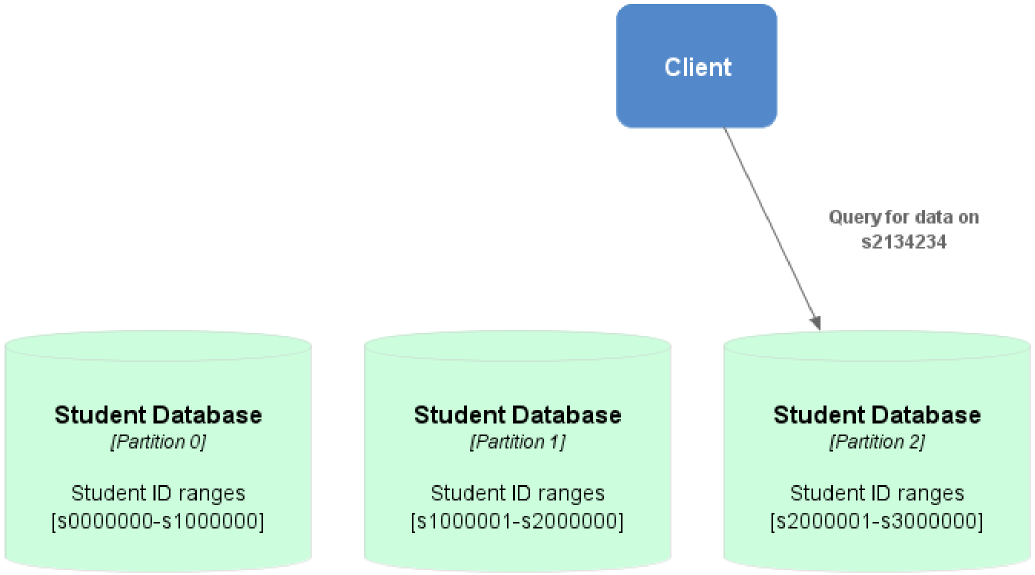
Query-Sensitive Load Balancer

Load balancer understands which queries should be forwarded to which node.


Client
Query for data on s2134234
Request aware load balancer
Forward request
Student Database
[Partition 0]
Student ID ranges [s0000000-s1000000]
Student Database
[Partition 1]
Student ID ranges [s1000001-s2000000]
Student Database
[Partition 2]
Student ID ranges [s2000001-s3000000]

Client-aware Queries

Place the responsibility on clients to choose the correct node.


Client
Query for data on s2134234
Student Database
[Partition 0]
Student ID ranges [s0000000-s1000000]
Student Database
[Partition 1]
Student ID ranges [s1000001-s2000000]
Student Database
[Partition 2]
Student ID ranges [s2000001-s3000000]

Summary

- *Partitioning* splits data across multiple nodes.

Summary

- *Partitioning* splits data across multiple nodes.
- Requires a *consistent method* to chose appropriate node.

Summary

- *Partitioning* splits data across multiple nodes.
- Requires a *consistent method* to chose appropriate node.
- Partitioning by *primary key* can create *skewing*.

Summary

- *Partitioning* splits data across multiple nodes.
- Requires a *consistent method* to chose appropriate node.
- Partitioning by *primary key* can create *skewing*.
- Partitioning by *hash* makes range queries less efficient.

Summary

- *Partitioning* splits data across multiple nodes.
- Requires a *consistent method* to chose appropriate node.
- Partitioning by *primary key* can create *skewing*.
- Partitioning by *hash* makes range queries less efficient.
- Three approaches to *routing requests*.

Disclaimer

We have ignored the *hard* parts of replication.

Question

How do we fix database scaling issues?

Question

How do we fix database scaling issues?

Answer

- Replication
- *Partitioning*
- Independent databases

Question

How do we fix database scaling issues?

Answer

- Replication
- Partitioning
- *Independent databases*

Summary

- Replications

Summary

- Replications
  - Leader-based, multi-leader, and leaderless

Summary

- Replications
  - Leader-based, multi-leader, and leaderless
  - Eventual consistency

Summary

- Replications
  - Leader-based, multi-leader, and leaderless
  - Eventual consistency
  - Write conflicts

Summary

- Replications
  - Leader-based, multi-leader, and leaderless
  - Eventual consistency
  - Write conflicts
- Partitioning

Summary

- Replications
  - Leader-based, multi-leader, and leaderless
  - Eventual consistency
  - Write conflicts
- Partitioning
  - Consistent method to pick nodes for data

Summary

- Replications
  - Leader-based, multi-leader, and leaderless
  - Eventual consistency
  - Write conflicts
- Partitioning
  - Consistent method to pick nodes for data
  - Avoiding skewing