An-Najah National University

Department of Computer Engineering

Distributed Operation Systems - 10636456

Lab #2 - Report

Project – part 2:Bazaar.com:
Replication, Caching and Consistency

Students:

Raghad Sabri 11924224

YaqoutSalameh11924020

In this project, we did everything together but we made Zoom and pushed the project to GitHub from one device most of the time.

In this phase of the project, we integrated the following features – Replication, Caching, and Consistency – into our Bazaar.com platform. In terms of replication, we generated replicas for both the order and catalog servers, enabling us to evenly distribute incoming requests between them to balance the load. In the realm of caching, we introduced a caching mechanism on the front end. This means that when a request is made, whether for information retrieval or search, we first check if we have previously handled this request by looking in the cache. If the request has been made before, we retrieve it from the cache (a "hit"). If it's the first occurrence of the request, we save it in the cache (a "miss") for future use. Finally, concerning consistency, when a write request is executed, resulting in a change in the database, we ensure that these changes are propagated across all servers. Additionally, we invalidate the cached value associated with the modified data to maintain data integrity.

The process for running the project remains unchanged from Part 1.

Here, we will present the results we obtained and substantiate them with visual evidence in the form of pictures.

Caching:

1- Before implementing the cache:

Before the implementation of caching, it was necessary to consistently retrieve results directly from the server for every read request, even if the same request had been made previously. This approach inevitably led to increased time overhead, thus reducing overall performance efficiency.

Response time for search before implementing cache:

```
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
1
Enter the topic you want to search for:
distributed systems
Response time: 2456 milliseconds
Performing search for topic: distributed systems
Search results:
[id='1', title='How to get a good grade in DOS in 40 minutes a day']
[id='2', title='RPCs for Noobs']
```

response time for info before implementing cache:

```
Enter the book id (1 to 4):
4
Displaying info for book with id: 4
Response time: 5 milliseconds
[id=4, topic='undergraduate school', title='Cooking for the Impatient Undergrad', price=30, quantity=5]
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 4):
4
Displaying info for book with id: 4
Response time: 4 milliseconds
[id=4, topic='undergraduate school', title='Cooking for the Impatient Undergrad', price=30, quantity=5]
```

2- After implementing cache:

Following the implementation of caching, the initial step is to consult the cache to determine if the desired result is already stored. If the result is found in the cache (a cache hit), retrieval is the quickest way. However, if the result is not in the cache (a cache miss), it necessitates fetching the result from the server, a process that takes longer time than cache hit and without cache.

Response time for search after implementing cache (hit-miss):

```
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
1
Enter the topic you want to search for:
distributed systems
Response time: 4696 milliseconds
Performing search for topic: distributed systems
Cache miss
Search results:
[id='1', title='How to get a good grade in DOS in 40 minutes a day']
[id='2', title='RPCs for Noobs']
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
1
Enter the topic you want to search for:
distributed systems
Response time: 1129 milliseconds
Performing search for topic: distributed systems
Cache hit
Search results:
[id='1', title='How to get a good grade in DOS in 40 minutes a day']
[id='2', title='RPCs for Noobs']
```

Response time for info after implementing cache (hit-miss):

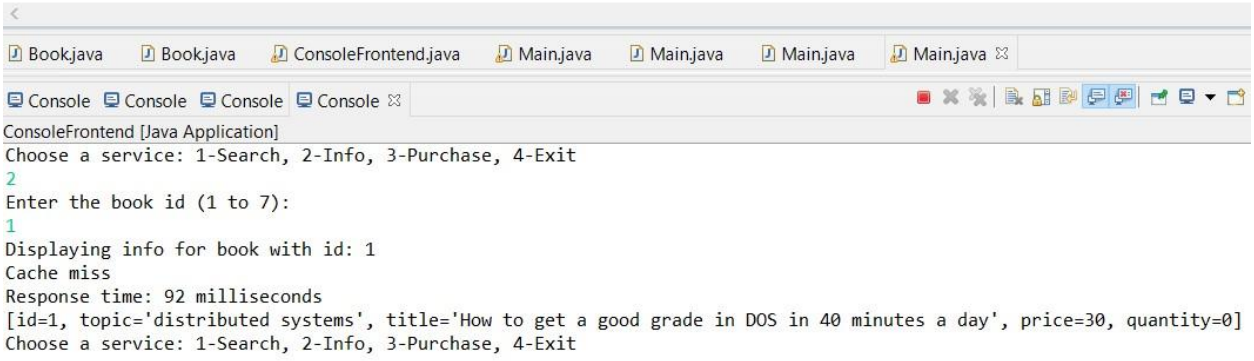
```
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 7):
1
Displaying info for book with id: 1
Cache miss
Response time: 7 milliseconds
[id=1, topic='distributed systems', title='How to get a good grade in DOS in 40 minutes a day', price=30, quantity=0]
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 7):
1
Displaying info for book with id: 1
Cache hit
Response time: 0 milliseconds
[id=1, topic='distributed systems', title='How to get a good grade in DOS in 40 minutes a day', price=30, quantity=0]
```

Replication:

We created duplicates of both the order and catalog servers, allowing us to distribute incoming requests evenly across these replicas, thereby achieving a balanced load distribution.

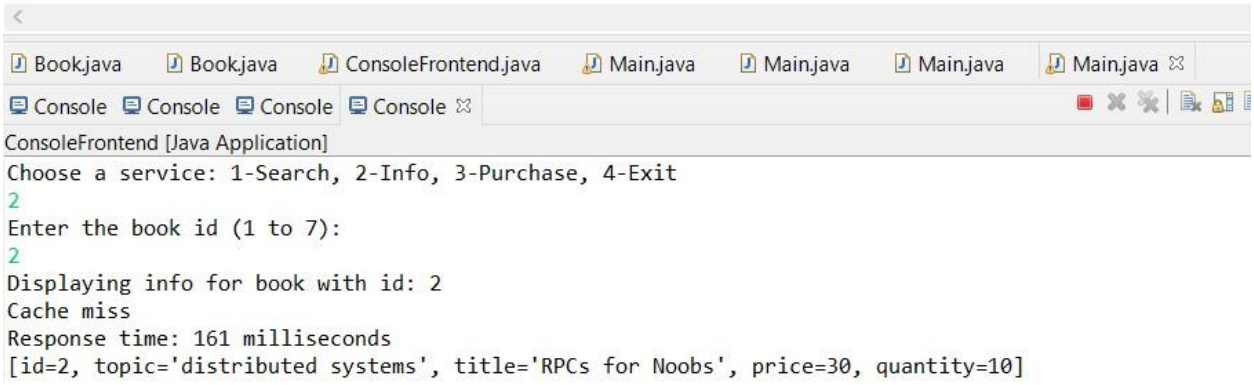
Replication primary server:

```
Connection to SQLite has been established.
Inside primary catalog server / info API
```



Replication secondary server:

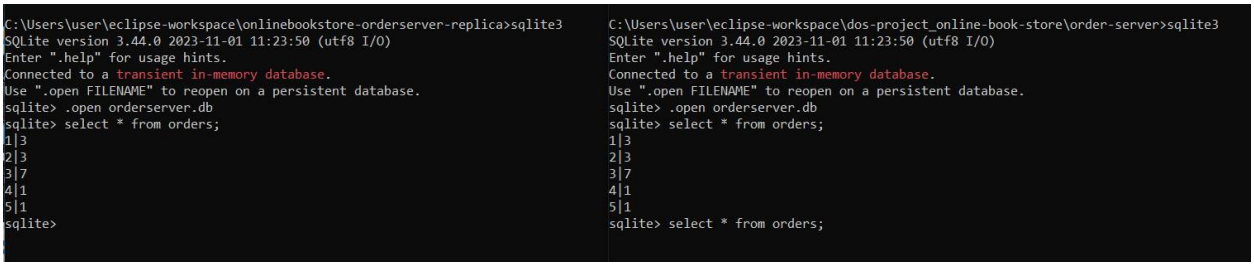
```
Connection to SQLite has been established.
Inside secondary catalog server / info API
```



Consistency:

When executing a written request that modifies the database, we take measures to replicate these changes across all servers. Furthermore, to preserve data integrity, we invalidate any cached values linked to the altered data. It's important to note that maintaining this consistency introduces additional time overhead.

Ensure order replica consistency (same database):



Write without implementing cache consistency:

```
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
3
Enter the book id (1 to 7):
5
Processing purchase for book with id: 5
Response time: 70 milliseconds
Purchase done successfully! The order id = 19
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
3
Enter the book id (1 to 7):
5
Processing purchase for book with id: 5
Response time: 75 milliseconds
Purchase done successfully! The order id = 20
```

Simple experiment - Cache Invalidation (Following the update of the book stack, the book's record in the cache becomes invalidated):

```
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 7):
5
Displaying info for book with id: 5
Cache miss
Response time: 12 milliseconds
[id=5, topic='new topic', title='How to finish Project 3 on time', price=30, quantity=9]
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 7):
5
Displaying info for book with id: 5
Cache hit
Response time: 0 milliseconds
[id=5, topic='new topic', title='How to finish Project 3 on time', price=30, quantity=9]
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
3
Enter the book id (1 to 7):
5
Processing purchase for book with id: 5
Response time: 100 milliseconds
Purchase done successfully! The order id = 17
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 7):
5
Displaying info for book with id: 5
Cache miss
Response time: 10 milliseconds
[id=5, topic='new topic', title='How to finish Project 3 on time', price=30, quantity=8]
```

Dockerizing the project:

```
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 7):
5
Displaying info for book with id: 5
Cache miss
Response time: 18 milliseconds
[id=5, topic='new topic', title='How to finish Project 3 on time', price=30, quantity=4]
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 7):
5
Displaying info for book with id: 5
Cache hit
Response time: 0 milliseconds
[id=5, topic='new topic', title='How to finish Project 3 on time', price=30, quantity=4]
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
3
Enter the book id (1 to 7):
5
Processing purchase for book with id: 5
Response time: 169 milliseconds
Purchase done successfully! The order id = 26
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
2
Enter the book id (1 to 7):
5
Displaying info for book with id: 5
Cache miss
Response time: 45 milliseconds
[id=5, topic='new topic', title='How to finish Project 3 on time', price=30, quantity=3]
Choose a service: 1-Search, 2-Info, 3-Purchase, 4-Exit
```

GitHub Repository for Online Book Store Project:

- Original Repository with Separate Branches for Each Server:
  https://github.com/YaqoutS/dos-project_online-book-store/tree/catalog-server
  In this repository, each server is in its own branch, allowing you to view commits and progression for each part of the code.

- Simplified Repository Structure for Easier Cloning and Testing:
  https://github.com/YaqoutS/dos-project_online-book-store_final.git
  This repository is well-structured, with all servers and code in the main branch. The structure mirrors local machines, ensuring correct paths in Docker files and Docker Compose.