Distributed Operating Systems

Bazar.com: A Multi-tier Online Book Store

Lama Ibrahim Hala Jabi

➔ Introduction:

In this lab, I restructured the online bookstore, Bazar.com, which I developed in Lab 1. Since the system finds it difficult to manage a larger amount of requests as a result of increased consumer demand, the objective is to enhance request processing time. In order to accomplish this, I implemented replication ,caching, and other design enhancements to produce an architecture that is more effective and scalable.

Additionally, this project intends to educate topics related to microservices, multi-tier web design, and Docker-based containerization.
The system changes performed to facilitate replication, caching, and consistency are described in full in the report that follows. I also go over the technical fixes put in place to improve scalability and lower latency.

➔ System Architecture and Design

○ New Architecture with Replication and Caching:

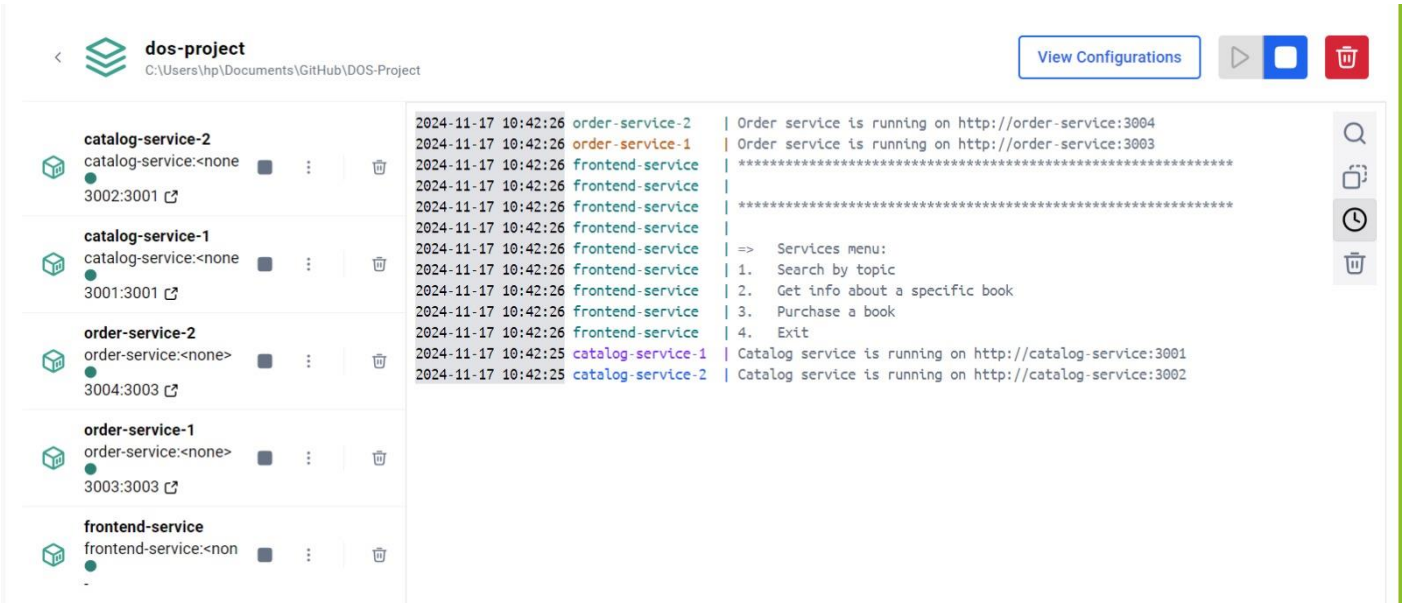
To enhance performance, we implemented a multi-layered architecture included the following components:

- **Front-End server with in-memory cache :**

To keep recently placed orders and catalog data that is often accessed, the front-end now has an in-memory cache.

- **Replicated Catalog and Order Services:**

To increase redundancy and speed up response times, I installed several clones of the catalog and order services to spread the load.



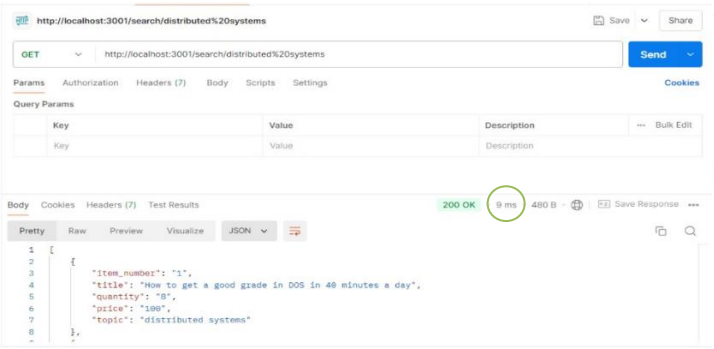
○ Implementation and Testing:

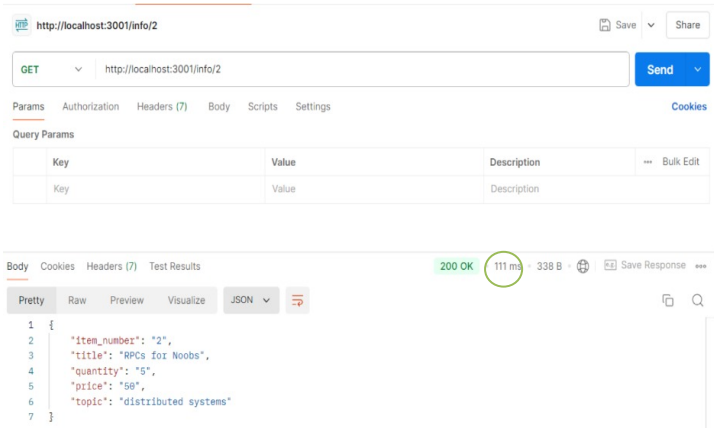
1. **In-Memory Caching**:
   We implement the code to manage an in-memory cache on the front-end server, improving response time by reducing database queries. The cache is designed as a key-value store to hold frequently requested items, like popular books and recent orders. It includes functions for retrieving data by key, storing new data, and invalidating outdated entries to maintain efficiency.

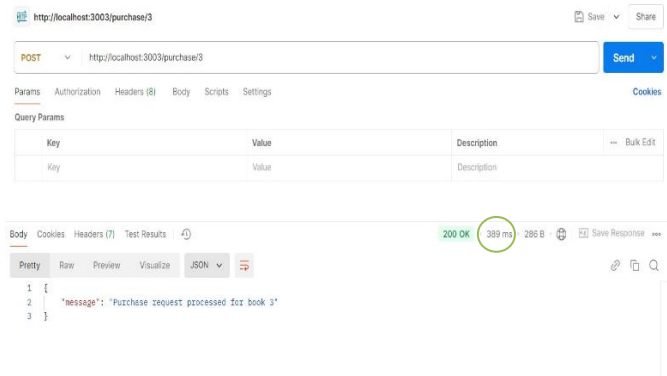
```
function getFromCache(key) {
  const entry = cache[key];
  if (entry) {
    return entry.data;
  }
  return null;
}

function setCache(key, data) {
  cache[key] = { data };
}

function invalidateCache(key) {
  if (cache[key]) {
    delete cache[key];
    console.log(`Cache invalidated for ${key}`);
  }
}
```

Implementing caching significantly reduced the average response time, as demonstrated by the measured results:

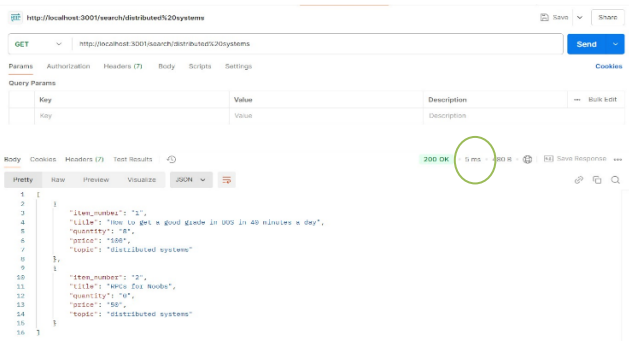
Without using cache

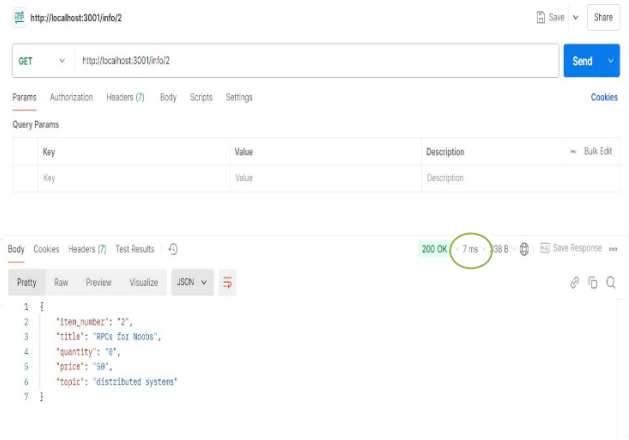


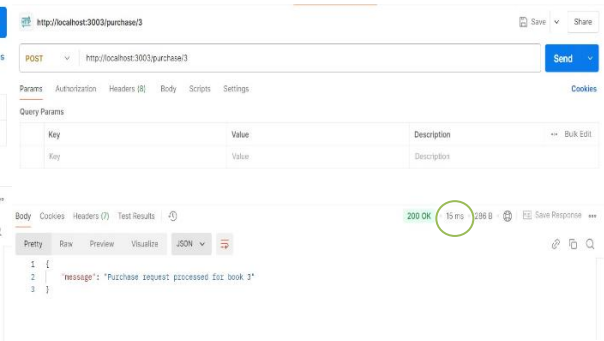




With using cache







The first time the topic is searched, the data is retrieved from the catalog service (cache miss). On the second search for the same topic, the data is fetched directly from the cache, demonstrating improved efficiency.

```
Write a number to choose an option: 1
Enter the needed topic: distributed systems
http://catalog-service-2:3002
cache miss
Books found:
```

(index)	item_number	title	quantity	price	topic
0	'1'	'How to get a good grade in DOS in 40 minutes a day'	'18'	'100'	'distributed systems'
1	'2'	'RPCs for Noobs'	'20'	'50'	'distributed systems'

```
1.   Search by topic
2.   Get info about a specific book
3.   Purchase a book
4.   Exit
Write a number to choose an option: 1
Enter the needed topic: distributed systems
Books found (from cache):
```

(index)	item_number	title	quantity	price	topic
0	'1'	'How to get a good grade in DOS in 40 minutes a day'	'18'	'100'	'distributed systems'
1	'2'	'RPCs for Noobs'	'20'	'50'	'distributed systems'

```
1.   Search by topic
2.   Get info about a specific book
3.   Purchase a book
4.   Exit
```

2. **Replication**:
   We implemented replication to enhance availability and reduce latency by creating multiple instances of the catalog and order services. The front-end server uses a round-robin strategy to distribute requests evenly across these replicas, ensuring balanced workload distribution and efficient handling of incoming requests.

```
const cache = {};

const catalogReplicas = [ Follow link (ctrl + click) ce-1:3001", "http://catalog-service-2:3002"];
const orderReplicas = ["http://order-service-1:3003", "http://order-service-2:3004"];

let catalogReplicaIndex = 0;
let orderReplicaIndex = 0;

function getNextCatalogReplica() {
  catalogReplicaIndex = (catalogReplicaIndex + 1) % catalogReplicas.length;
  console.log(catalogReplicas[catalogReplicaIndex]);
  return catalogReplicas[catalogReplicaIndex];
}

function getNextOrderReplica() {
  orderReplicaIndex = (orderReplicaIndex + 1) % orderReplicas.length;
  console.log(orderReplicas[orderReplicaIndex]);
  return orderReplicas[orderReplicaIndex];
}
```

The workload is distributed across two catalog replicas: the topic search is handled by `http://catalog-service-2:3002`, and the book info request by `http://catalog-service-1:3001`, ensuring balanced performance.

```
Write a number to choose an option: 1
Enter the needed topic: undergraduate school
http://catalog-service-2:3002
cache miss
Books found:
```

(index)	item_number	title	quantity	price	topic
0	'3'	'Xen and the Art of Surviving Undergraduate School'	'18'	'150'	'undergraduate school'
1	'4'	'Cooking for the Impatient Undergrad'	'20'	'20'	'undergraduate school'

```
1.   Search by topic
2.   Get info about a specific book
3.   Purchase a book
4.   Exit
Write a number to choose an option: 2
Enter book number: 3
http://catalog-service-1:3001
Book info:
```

(index)	item_number	title	quantity	price	topic
0	'3'	'Xen and the Art of Surviving Undergraduate School'	'18'	'150'	'undergraduate school'

```
1.   Search by topic
2.   Get info about a specific book
3.   Purchase a book
4.   Exit
Write a number to choose an option:
```