Distributed operating systems

A Multi-tier Online Book Store part2

Instructor:

Samer Arandi

Students:

Hala okal

Farah toqan

Introduction:

In this part, we add many techniques to make the performance better in this project.

We used 3 ubuntu VMs in the previous part

First, we add an in-memory cache that caches the results of recent requests to the order and catalog servers.

We use the library ('node-cache') to allow us make the in-memory cache

And we make it using this statement (const cache = new NodeCache({ stdTTL: 500});)

Second, we deal with replication of catalog and order servers, so we make a load balancing algorithm in front end server that takes each incoming request and sends it to one of the replicas, we use Round-Robin algorithm.

The idea of Round-robin is to start a new server every time a request is made, and we make it like in the image.

And to calculate the time taken for purchase and search requests we use ('perf_hooks') library and calculate the time from the start of the function to its end, and we make it like in the image.

```
app.get('/CATALOG_WEBSERVICE_IP/search/:itemName', async (req, response) => {


    const url = slaveServers[currentServerIndex] + `/CATALOG_WEBSERVICE_IP/find/${req.params.itemName}`;
    try {
        const start = performance.now();
        console.log('req.params.itemName' + req.params.itemName);
        const data = await cachedRequest(url, req.params.itemName);
        if (data === '0') {
            info = 'the book is not found';
        } else {
            info = data;
        }
        console.log('info = ', info);
        response.send(info);

        currentServerIndex = (currentServerIndex + 1) % slaveServers.length;
        const end = performance.now();
        const elapsedTime = end - start;

        console.log('Time taken ', elapsedTime, 'milliseconds');
    } catch (error) {
        console.log(error);
        response.status(500).send('Internal Server Error');
    }
});
```

When we make an order from the front-end server, it sends it to the order server, then the order server sends it to the catalog to edit on the database (dec the quantity in catalog.csv file and add new order in order.csv file).

We used this function:

```
async function sendOrderToCatalogServer(order) {
    try {
        const catalogServerUrl = 'http://10.0.2.4:4000';
        const catalogUpdateUrl = `${catalogServerUrl}/CATALOG_WEBSERVICE_IP/updateCatalog`;

         axios.post(catalogUpdateUrl, order).then((response) => {
            console.log('data sent t0 catal0g');
            console.log('resp0nse',response.data);

        }).catch ((error)=> {
            console.error('Error sending order data to catalog server:', error);
        });

        console.log('Order data sent to catalog server:', order);
    } catch (error) {
        console.error('Error sending order data to catalog server:', error);
    }
}
```

Here are the results when we make orders and calculate the time taken 4 times

This is the result when we send an order request for the first time

```
Server is running on port 4001
{ hits: 0, misses: 0, keys: 0, ksize: 0, vsize: 0 }
cacheKey1
cachedDataundefined
info =  {"orderId":61,"itemName":"How to get a good grade in DOS in 40 minutes a day","itemPrice":"100
"}
Time taken  715.1981630008668 milliseconds
{ hits: 0, misses: 1, keys: 1, ksize: 1, vsize: 96 }
cacheKey1
cachedData{"orderId":61,"itemName":"How to get a good grade in DOS in 40 minutes a day","itemPrice":"1
00"}
Cache hit: 1
info =  {"orderId":61,"itemName":"How to get a good grade in DOS in 40 minutes a day","itemPrice":"100
"}
Time taken  273.7759009990841 milliseconds
```

This is the result when we send an order request for the second time

```
{ hits: 1, misses: 1, keys: 1, ksize: 50, vsize: 132 }
cacheKeyRPCs for Noobs
cachedDataundefined
info =  [{"id":"2","price":"100","title":"RPCs for Noobs","quantity":0,"topic":"distributed systems"}]
Time taken  139.79035400040448 milliseconds
req.params.itemNameRPCs for Noobs
{ hits: 1, misses: 2, keys: 2, ksize: 64, vsize: 226 }
cacheKeyRPCs for Noobs
cachedData[{"id":"2","price":"100","title":"RPCs for Noobs","quantity":0,"topic":"distributed systems"
}]
Cache hit: RPCs for Noobs
info =  [{"id":"2","price":"100","title":"RPCs for Noobs","quantity":0,"topic":"distributed systems"}]
Time taken  40.25714999996126 milliseconds
```

This is the result when we send an order request for the third time

```
{ hits: 2, misses: 2, keys: 2, ksize: 64, vsize: 226 }
cacheKeyXen and the Art of Surviving Undergraduate School
cachedDataundefined
info =  [{"id":"3","price":"120","title":"Xen and the Art of Surviving Undergraduate School","quantity
":5,"topic":"undergraduate school"}]
Time taken  111.64628499932587 milliseconds
req.params.itemNameXen and the Art of Surviving Undergraduate School
{ hits: 2, misses: 3, keys: 3, ksize: 113, vsize: 356 }
cacheKeyXen and the Art of Surviving Undergraduate School
cachedData[{"id":"3","price":"120","title":"Xen and the Art of Surviving Undergraduate School","quanti
ty":5,"topic":"undergraduate school"}]
Cache hit: Xen and the Art of Surviving Undergraduate School
info =  [{"id":"3","price":"120","title":"Xen and the Art of Surviving Undergraduate School","quantity
":5,"topic":"undergraduate school"}]
Time taken  40.014991998672485 milliseconds
```

This is the result when we send an order request for the fourth time

```
{ hits: 3, misses: 3, keys: 3, ksize: 113, vsize: 356 }
cacheKeyCooking for the Impatient Undergrad
cachedDataundefined
info =  [{"id":"4","price":"50","title":"Cooking for the Impatient Undergrad","quantity":16,"topic":"u
ndergraduate school"}]
Time taken  84.4198679998517 milliseconds
req.params.itemNameCooking for the Impatient Undergrad
{ hits: 3, misses: 4, keys: 4, ksize: 148, vsize: 472 }
cacheKeyCooking for the Impatient Undergrad
cachedData[{"id":"4","price":"50","title":"Cooking for the Impatient Undergrad","quantity":16,"topic":
"undergraduate school"}]
Cache hit: Cooking for the Impatient Undergrad
info =  [{"id":"4","price":"50","title":"Cooking for the Impatient Undergrad","quantity":16,"topic":"u
ndergraduate school"}]
Time taken  46.568019000813365 milliseconds
```

This is the result when we send a search request for the first time

```
{ hits: 0, misses: 0, keys: 0, ksize: 0, vsize: 0 }
cacheKeyHow to get a good grade in DOS in 40 minutes a day
cachedDataundefined
info =  [{"id":"1","price":"100","title":"How to get a good grade in DOS in 40 minutes a day","quantit
y":-21,"topic":"distributed systems"}]
Time taken  374.73604299873114 milliseconds
req.params.itemNameHow to get a good grade in DOS in 40 minutes a day
{ hits: 0, misses: 1, keys: 1, ksize: 50, vsize: 132 }
cacheKeyHow to get a good grade in DOS in 40 minutes a day
cachedData[{"id":"1","price":"100","title":"How to get a good grade in DOS in 40 minutes a day","quant
ity":-21,"topic":"distributed systems"}]
Cache hit: How to get a good grade in DOS in 40 minutes a day
info =  [{"id":"1","price":"100","title":"How to get a good grade in DOS in 40 minutes a day","quantit
y":-21,"topic":"distributed systems"}]
Time taken  52.13630600087345 milliseconds
```

This is the result when we send a search request for the second time

```
{ hits: 1, misses: 1, keys: 1, ksize: 1, vsize: 96 }
cacheKey2
cachedDataundefined
info =  {"orderId":64,"itemName":"RPCs for Noobs","itemPrice":"70"}
Time taken  547.8916210010648 milliseconds
{ hits: 1, misses: 2, keys: 2, ksize: 2, vsize: 155 }
cacheKey2
cachedData{"orderId":64,"itemName":"RPCs for Noobs","itemPrice":"70"}
Cache hit: 2
info =  {"orderId":64,"itemName":"RPCs for Noobs","itemPrice":"70"}
Time taken  119.4409110005945 milliseconds
```

This is the result when we send a search request for the third time

```
{ hits: 2, misses: 2, keys: 2, ksize: 2, vsize: 155 }
cacheKey3
cachedDataundefined
info =  {"orderId":67,"itemName":"Xen and the Art of Surviving Undergraduate School","itemPrice":"120"
}
Time taken  360.8598250001669 milliseconds
{ hits: 2, misses: 3, keys: 3, ksize: 3, vsize: 250 }
cacheKey3
cachedData{"orderId":67,"itemName":"Xen and the Art of Surviving Undergraduate School","itemPrice":"12
0"}
Cache hit: 3
info =  {"orderId":67,"itemName":"Xen and the Art of Surviving Undergraduate School","itemPrice":"120"
}
Time taken  153.28570299968123 milliseconds
```

This is the result when we send a search request for the fourth time

```
{ hits: 3, misses: 3, keys: 3, ksize: 3, vsize: 250 }
cacheKey4
cachedDataundefined
info =  {"orderId":70,"itemName":"Cooking for the Impatient Undergrad","itemPrice":"80"}
Time taken  382.50761299952865 milliseconds
{ hits: 3, misses: 4, keys: 4, ksize: 4, vsize: 330 }
cacheKey4
cachedData{"orderId":70,"itemName":"Cooking for the Impatient Undergrad","itemPrice":"80"}
Cache hit: 4
info =  {"orderId":70,"itemName":"Cooking for the Impatient Undergrad","itemPrice":"80"}
Time taken  205.56312000006437 milliseconds
```

The table:

	Search without cache	Search with cache	purchase without cache	purchase with cache
Item1	274.736	52.136	715.198	273.775
Item2	139.790	40.257	547.891	119.440
Item3	111.646	40.014	360.859	153.285
Item4	84.419	46.568	382.507	205.563

We conclude that in the first time we make an order or a search request it take a very big time because of the cache miss, but then the data stored in the cache and in the second time we don’t need to go to the memory