An-Najah National University

Computer Engineering department

Distributed and Operating Systems Course

Web Microservices HW#3 - Online BookStore Improvements

Moath AbuSada

15 Dec 2020

Task

Our online Bazar become famous, so product department request from development team to do the following:

1. Added new three books under the “spring sale” topic name
2. Rearchitecting Bazar.com's online store to handle this higher workload by support the two following techniques:
    a. Replicate the catalog and order servers on multiple machines
    b. Implement loading balancing algorithm
    c. Develop caching techniques to improve request processing latency

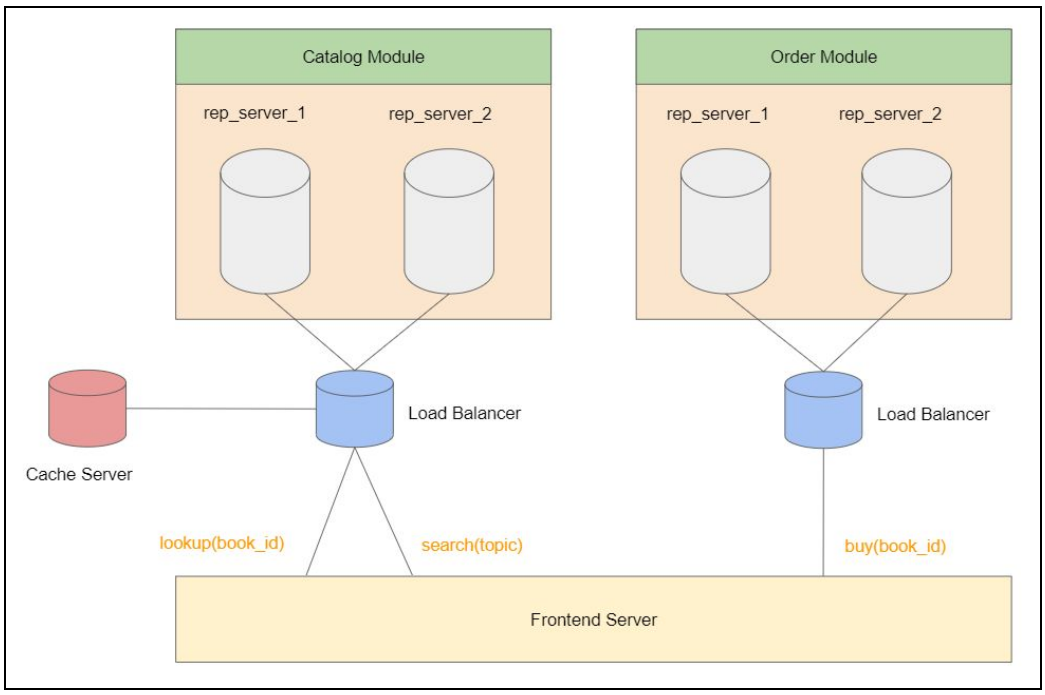
Solution

System Design

The new system design supported by four modules

1. Catalog module : contains all catalog server replicas in addition to load balancer server to manage routing between replicas
2. Order module : contains all order server replicas in addition to load balancer server to manage routing between replicas
3. Cache server : interacted with load balancer of Catalog module, and provide high speed caching for lookups
4. Frontend server : one server interacted with both Load balancers

System Model Diagram



System Components Description

Regarding **Catalog and Order Server Replicas**, each replica is a live virtual machine run with ubuntu server OS with apache server and has its own Database.

Regarding the **Client Server** is a basic html page that makes interaction with the load balancer of catalog replicas and load balancer of order replicas, by ajax and http apis. So the client page can be located on a live server or can be run on a local client pc.

Regarding the **Load Balancer,** load balancer is a server located at each cluster of replicas, so catalog replicas have their own load balancer server, and orders replicas have their own load balancer server. We implement the load balancer algorithm based on the least loaded protocol. So at each coming request from frontend, we forward the request to the replica that has the least number of current tasks. We maintain the state of replicas by a standalone Databases which handles replica ip address, name, current task, availability status.

Regarding **Cache Server**, the cache server interacts directly with the Load balancer server of catalog replicas. Cache server stores the book lookups as a standalone file for each book, and inside each file we store the book info as a JSON object. At each lookup request coming from frontend to Load Balancer server of catalog replicas, we check the cache server if contains the looked up book, if found it we return the book info directly to the client, and if not found the book lookup at cache we forward the request to one replicas based on load balancing strategy, and after that we place the book into cache.

How Load Balancers Work ?

When a new request comes from frontend, LB check replicas table and get least load replica and the replica availability status should be true.

Then we get the full path for the new replica and store it in two variables and assign a new task for the targeted replica to ready it for the coming task. If there is no available replica we respond with no available server.

```php
// replicas managment

$balancer_db = new DataBaseInstance('localhost', 'admin', '11223344', 'bookstore_db_catalog_balancer', '3306');


$query = $balancer_db->query("SELECT * FROM replicas WHERE `status` = 1 ORDER BY current_tasks ASC LIMIT 1");

if ($query->num_rows) {
    $destination_ip = $query->row['ip'];
    $destination_name = $query->row['name'];
    $balancer_db->query("UPDATE replicas SET current_tasks = (current_tasks + 1) WHERE `name` = '" . $destination_name . "'");
} else {
    header("HTTP/1.1 404 No Available Server");
    exit();
}
```

On the operation section, we request the operation from the targeted replica. After replica end the assigned task we decrease the current tasks for this replicas on LB Database

```php
// server operations

if ($uri[3] == 'search') {

    $url = "http://" . $destination_ip . "/" . $destination_name . "/search.php?topic=" . $uri[4];
    $curl = curl_init($url);
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $resp = curl_exec($curl);
    curl_close($curl);

    echo $resp;

    $balancer_db->query("UPDATE replicas SET current_tasks = (current_tasks - 1) WHERE `name` = '" . $destination_name . "'");
```

How does cache work ?

If a book lookup request coming from frontend to balancer, balancer search on cache server for this book lookup, if found respond to frontend directly with book data, if book lookup not found on cache, LB forward the request to one replicas, and after operation ends and data returned to frontend, LB update cache server with the new book lookup for a future operation.

Please note that on replica management, we assign the task for targeted replica even before check on the cache (for lookup operation), because the cache check is especially for some operations and not for all, so after enter lookup operation and check on cache, if book lookup found the look we don’t forget to decrease the current tasks for targeted replica. This approach is used today and it helps in reduce the time of response because not all operations need a cache management.

```php
} elseif ($uri[3] == 'lookup') {

    // check the cache first

    $cache_result = $cache_server->get("lookup." . $uri[4]);

    if ($cache_result) {

        $cache_result['from_cache'] = true;
        echo json_encode($cache_result);

    } else {

        // if the lookup not found on cache, read it from server

        $url = "http://" . $destination_ip . "/" . $destination_name . "/lookup.php?book_id=" . $uri[4];
        $curl = curl_init($url);
        curl_setopt($curl, CURLOPT_URL, $url);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        $resp = curl_exec($curl);
        curl_close($curl);

        echo $resp;

        $balancer_db->query("UPDATE replicas SET current_tasks = (current_tasks - 1) WHERE `name` = '" . $destination_name . "'");


        // cache update (add lookup)
        $cache_server->set("lookup." . $uri[4], json_decode($resp, true));
    }
```

How do we manage Databases Consistency ?

As we know after adding a new order, the quantity will decrease. So we manage the consistency between all replicas by updating them with the new data after any update_quantity operation occurred.

```php
} elseif ($uri[3] == 'update_quantity') {

    $url = "http://" . $destination_ip . "/" . $destination_name . "/update_quantity.php?book_id=" . $uri[4];
    $curl = curl_init($url);
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $resp = curl_exec($curl);
    curl_close($curl);

    echo $resp;

    $balancer_db->query("UPDATE replicas SET current_tasks = (current_tasks - 1) WHERE `name` = '" . $destination_name . "'");


    // All replicas DB sync
    syncCatalogServers($destination_name, $uri[4]);
```

```php
// internal DBs syncing

function syncCatalogServers($source, $book_id)
{

    $replica_db = new DataBaseInstance('localhost', 'admin', '11223344', "bookstore_db_catalog_" . $source, '3306');

    $sql = "SELECT quantity FROM books WHERE book_id = '" . $book_id . "'";

    $query = $replica_db->query($sql);

    $quantity = $query->row['quantity'];


    $balancer_db2 = new DataBaseInstance('localhost', 'admin', '11223344', 'bookstore_db_catalog_balancer', '3306');

    $query = $balancer_db2->query("SELECT * FROM replicas WHERE `status` = 1 AND name != '" . $source . "'");

    foreach ($query->rows as $result) {

        $replica_db2 = new DataBaseInstance('localhost', 'admin', '11223344', "bookstore_db_catalog_" . $result['name'], '3306');

        $sql = "UPDATE books SET quantity = '" . (int)$quantity . "' WHERE book_id = '" . (int)$book_id . "'";

        $query = $replica_db2->query($sql);
    }
}
```

Also according to order replicase databases consistency, we update all DBs with the new added order

```php
// server operations

if ($uri[3] == 'buy') {

    $url = "http://" . $destination_ip . "/" . $destination_name . "/buy.php?book_id=" . $uri[4];
    $curl = curl_init($url);
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $resp = curl_exec($curl);
    curl_close($curl);

    echo $resp;

    $balancer_db->query("UPDATE replicas SET current_tasks = (current_tasks - 1) WHERE `name` = '"


    // All replicas DB sync
    syncOrderServers($destination_name, $uri[4]);
```

```php
// internal syncing

function syncOrderServers($source, $book_id){

    $balancer_db2 = new DataBaseInstance('localhost', 'admin', '11223344', 'bookstore_db_order_balancer', '3306');

    $query = $balancer_db2->query("SELECT * FROM replicas WHERE `status` = 1 AND name != '" . $source . "'");

    foreach ($query->rows as $result) {

        $replica_db = new DataBaseInstance('localhost', 'admin', '11223344', "bookstore_db_order_" . $result['name'], '3306');

        $sql = "INSERT INTO orders SET book_id = '" . $book_id . "', order_date = NOW()";

        $query = $replica_db->query($sql);

    }
}
```

How do we manage Cache Consistency ?

After each update_quantity operation, the book info on cache will be different than original info on DB, so after each update_quantity operation, we search on cache if the related book lookup on cache, if there we delete it.

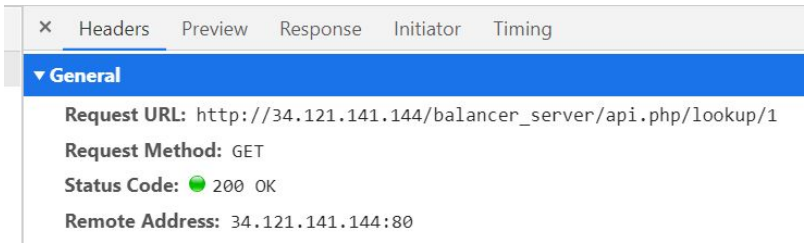
```php
} elseif ($uri[3] == 'update_quantity') {

    $url = "http://" . $destination_ip . "/" . $destination_name . "/update_quantity.php?book_id=
    $curl = curl_init($url);
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $resp = curl_exec($curl);
    curl_close($curl);

    echo $resp;

    $balancer_db->query("UPDATE replicas SET current_tasks = (current_tasks - 1) WHERE `name` = '

    // All replicas DB sync
    syncCatalogServers($destination_name, $uri[4]);

    // cache update (remove lookup)
    $cache_server->delete("lookup." . $uri[4]);
```

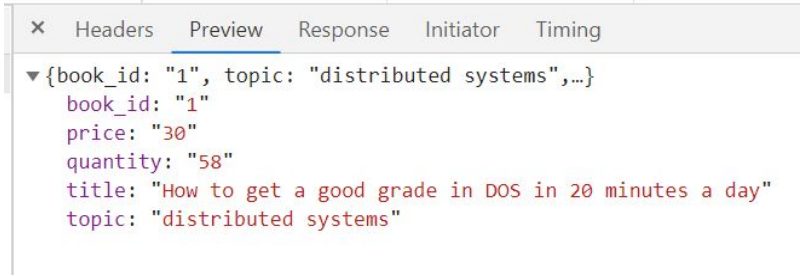
Cache Performance measurement

We do a lookup for book_id = 1, we know that the book is not on cache. After we requested it for the first time the LB returned it from DB, but if we request it the second time the LB will return it from the cache server. The testing done under the same circumstances. Note that we have a line code (just for testing) appended to JSON to distinguish if data returns from DB or from cache server.

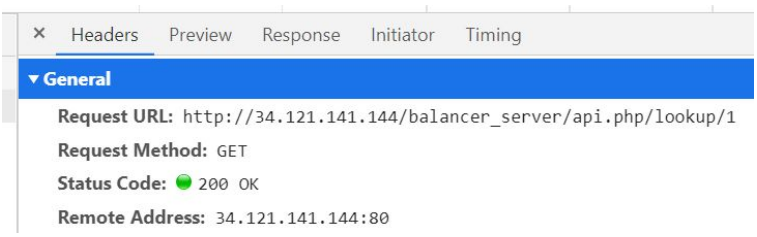
```php
if ($cache_result) {

    $cache_result['from_cache'] = true;  // just for testing
    echo json_encode($cache_result);
```

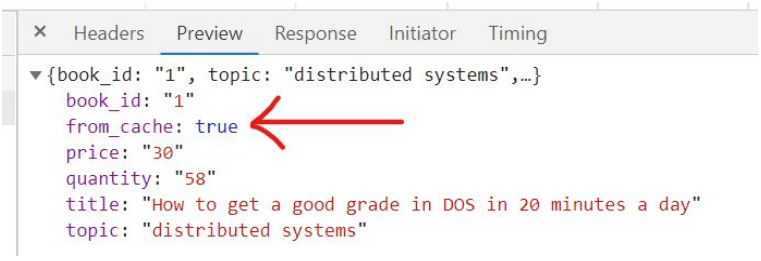
<u>First request result:</u>

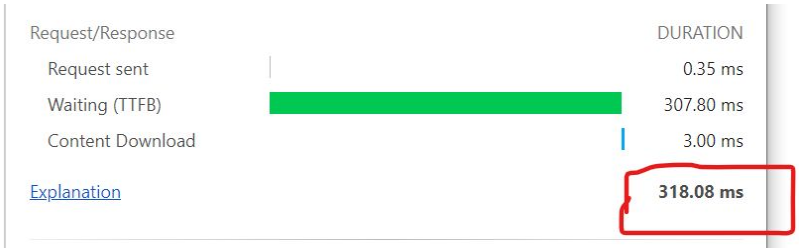




Second request result:







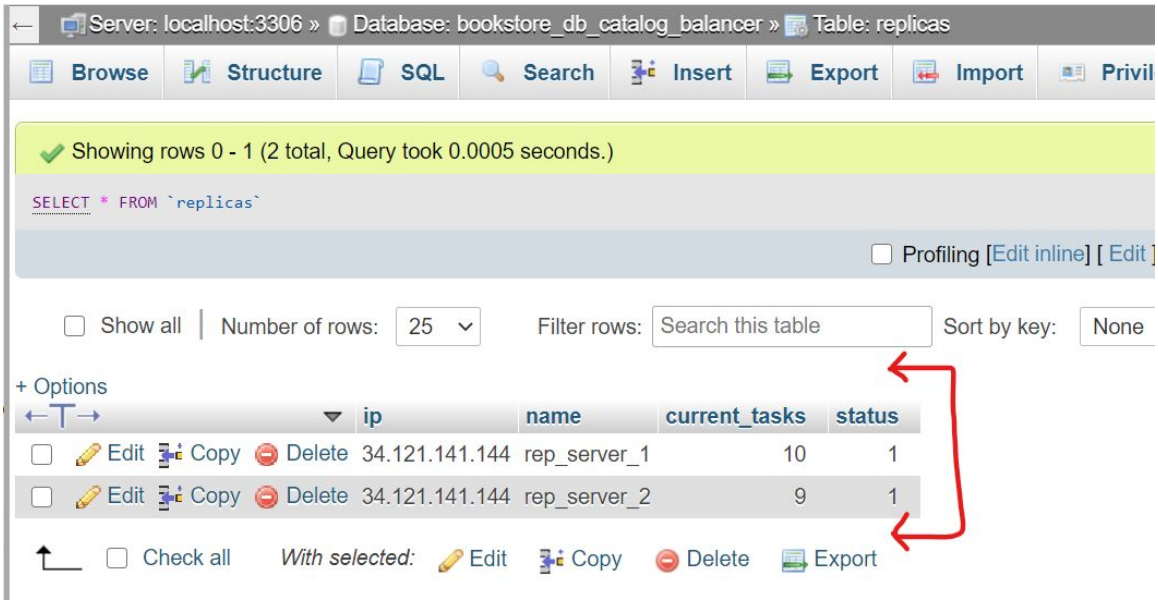
Improvement Percentage = (old_time - new_time) / new_time X 100%

(462 - 318) / 318 X 100% = 45%

That means, with using cache,the response time improved by approx 45% compared without using cache.

The Power of scalability “Scale up on one minute”

Our new system design is supported to scale up horizontally without changing any implementation, just add the replica record to replicas table on Load Balancer Database by entering replica ip address, name and availability status.



Github Repository

Link : https://github.com/moathabusada/doshw2.git

Content:

- Project PDF Report
- frontend folder contain client page
- catalog_cluster folder, contains each replica server folder, LB server folder, and cache server folder
- order_cluster folder, contains each replica server folder, LB server folder.
- Each replica and LB Database scheme attached on each server folder in “db_schema” sub folder

The End