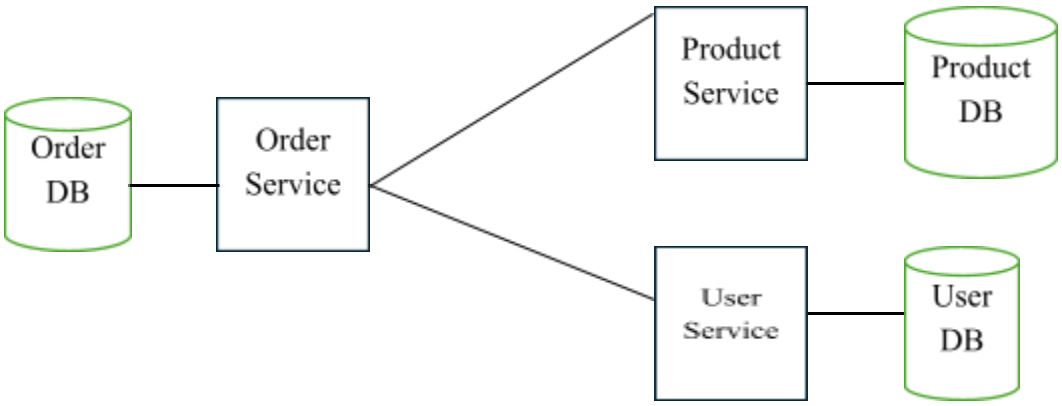
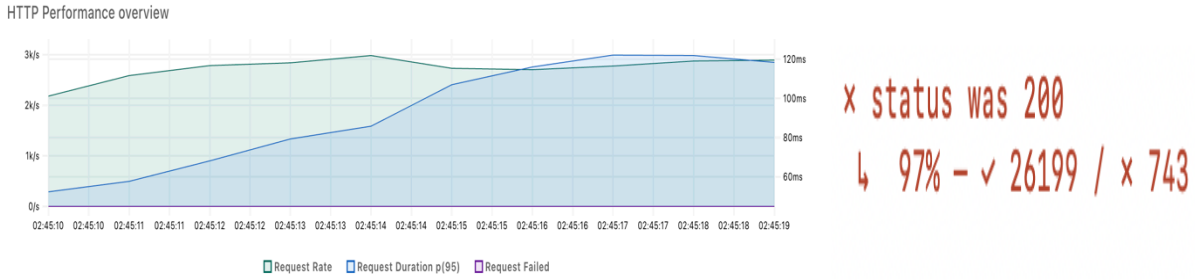
In our test, 10 threads were set up using Python to send a post request to the server to create 1000 users and products and 5500 orders and record the time to completion. Then we used the K6 load test script to test the server performance. Find the server performance bottleneck by setting up 200 threads to send a large number of Get requests to the server for 10 seconds.

In base systems, we use the order server to receive all the requests, and according to the type of request to choose to forward to the product/user server or their processing, each server has an SQLite database.



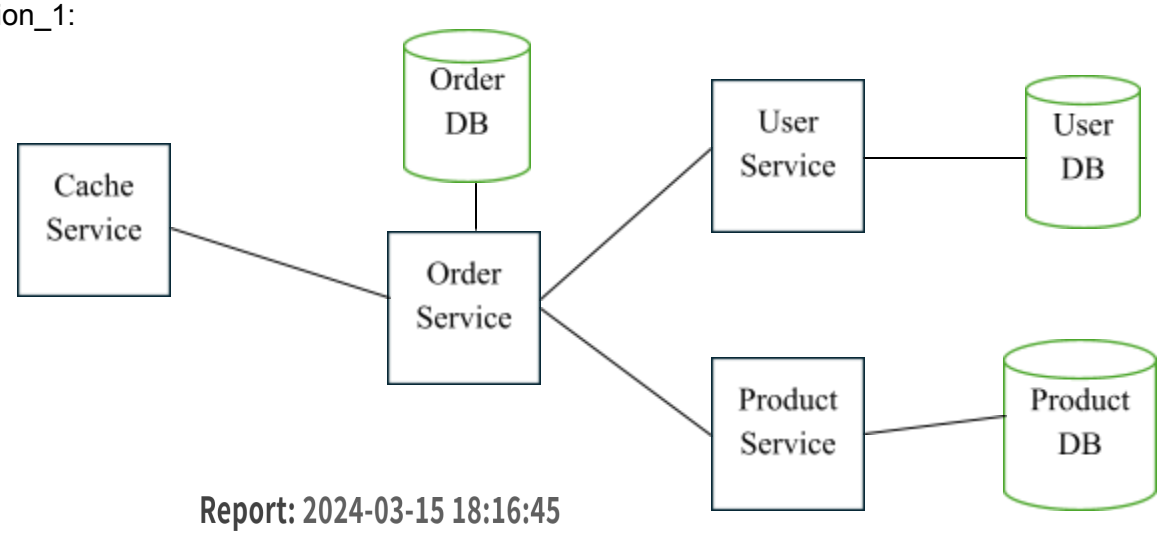


By running the test, we found that base systems take about 60.37 seconds to process all the POST requests. in a 10-second continuous test, we can see that the server can handle about 2000 Get requests per second.

We can see that the server is much slower in processing a post request than a get request, this is because the server needs to communicate with SQLite to process the request, and write operations usually take more time than reads.

Based on the test results, we decided to add a cache server in front of the server to temporarily store the data using a hash map. When a request is received, the cache server will keep the processed data in the cache, and then every once in a while, it will send the data to the Order service and it will write the data to SQLite. To speed up the response of the server.

Version_1:

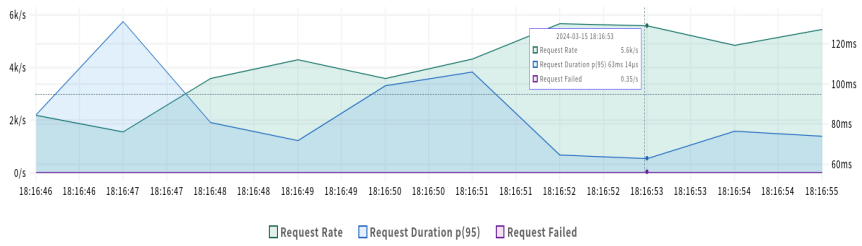


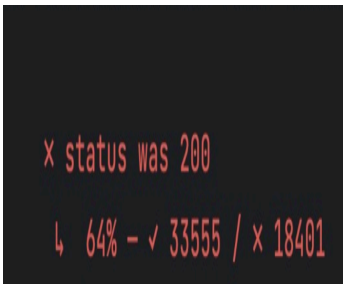
Report: 2024-03-15 18:16:45

verview

s chapter provides an overview of the most important metrics of the test run. Graphs plot the value of metrics over time.

TTP Performance overview



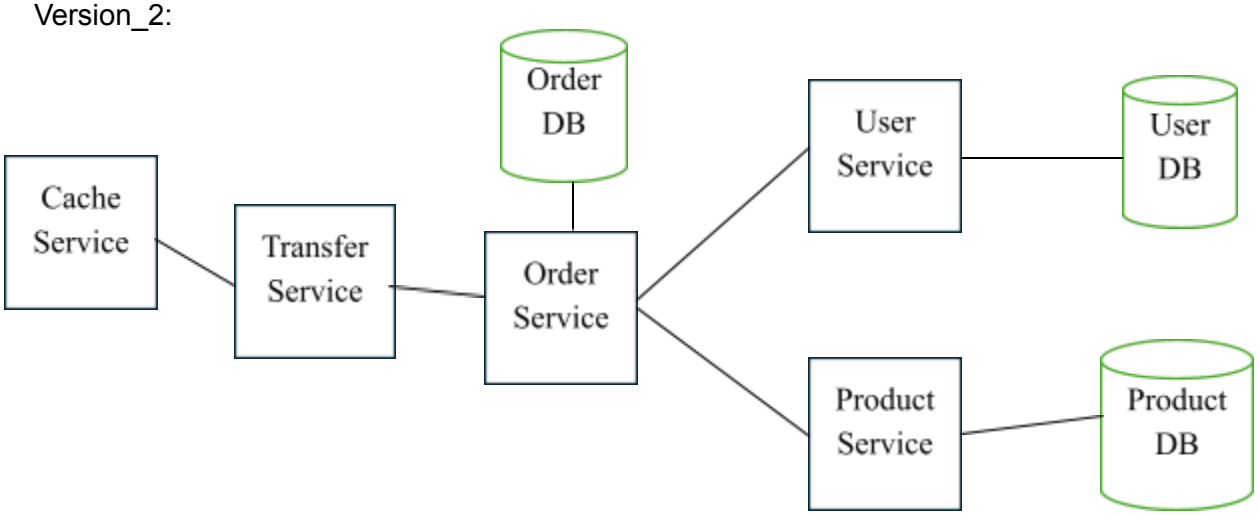


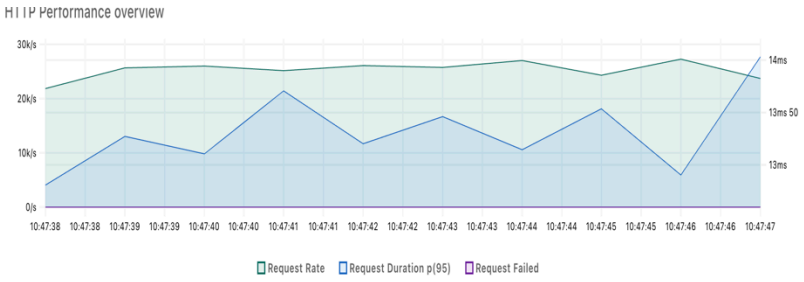
By running the test, we found that base systems take about 6.34 seconds to process all the POST requests. in a 10-second continuous test, we can see that the server can handle about 5000 Get requests per second.

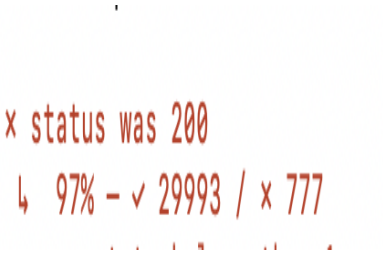
After adding the caching service, the effect is immediate, compared to the original 60.37-second Post requests processing speed is now only 6 seconds, this is because the cache relative to the storage speed is much faster. get request processing speed has also risen to more than 5000 per second. However, it can be seen that only 64% of the requests returned correct results, which is not normal.

We speculate that it may be that the Cache service then sends data to the Order Service while conducting the test. The Order Service is slow to process the Post request. During this time the Cache service needs to wait for the Order Service, which is a waste of performance. So we decided to add an extra transfer service between the Cache service and Order Service. The transfer service waits for the Order Service to process the Post request.

Version_2:





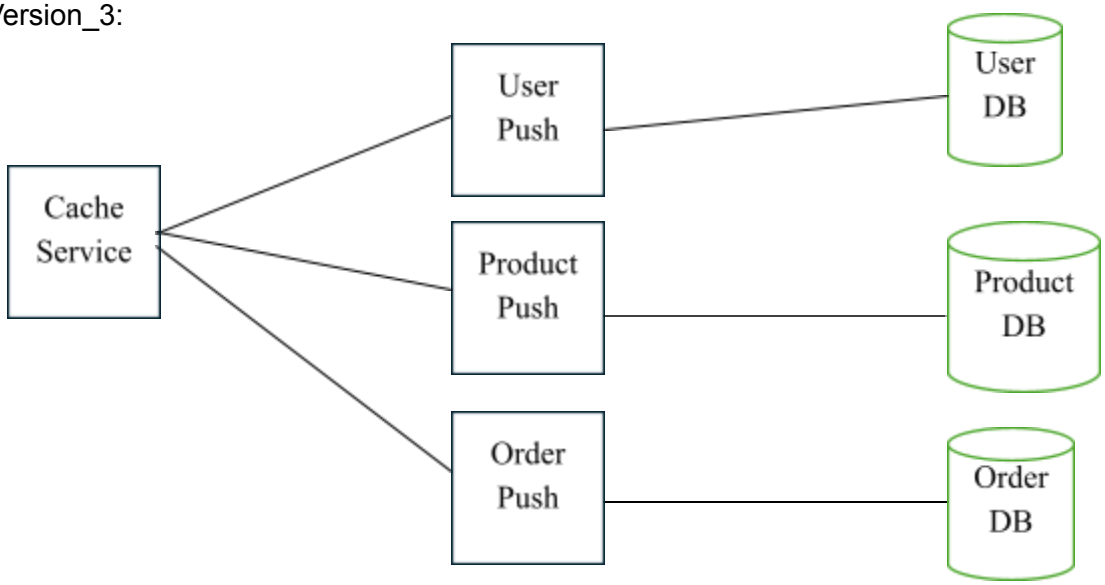


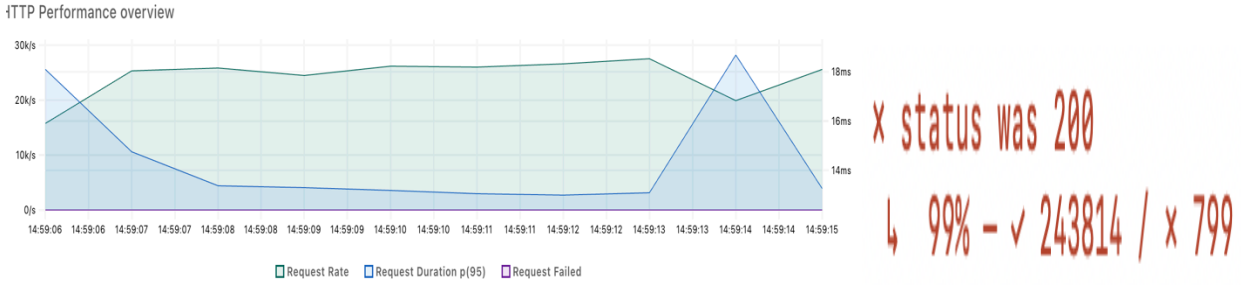
By running the test, we found that base systems take about 6.34 seconds to process all the POST requests. But this time in a 10-second continuous test, the server can handle about 20000 Get requests per second. This is because now the Cache server doesn't have to wait for the Order server.

However, we realized that our architecture seemed a bit too complex and wasteful of resources. For example, in Cache Service, we have validated all the requests, but in Order Service, we repeat the validation again. In fact, we only need to write the data to SQLite.

We removed the Transfer Service and dropped the old Order, User, Product Service and created Order, User, Product Push that can accept all commands at once and communicate with their respective SQLite. We also integrated the distribution functionality into the Cache Service, and we also need to detect changes in the hash map data in every persistence period before sending a command to update the SQLite to the Order, User, and Product Push.

Version_3:





By running the test, we found that base systems take about 5.82 seconds to process all the POST requests. In a 10-second continuous test, the server still can handle over 20000 Get requests per second. This time, we have dramatically simplified our model to improve server resource utilization while maintaining performance.

Version	Performance	Defect	Optimize
Original	(Post Request): 60.37 for 7500 Request (Get Request): 2000/s	Handel every request from SQLite is too slow.	Adding a Cache Server
1	(Post Request): 6.34 for 7500 Request (Get Request):5000/s	There is a performance improvement but the need to wait for the Order service results in not being able to process the request well.	Adding a Transfer Server between the Cache Server and the Order Server
2	(Post Request): 6.34 for 7500 Request (Get Request):20000/s	Performance is improved but the model is too complex resulting in a waste of resources.	Remove redundant functionality and refactor the service for model simplification.
3	(Post Request):5.82 for		

	7500 Request (Get Request):20000/s		