The University of Melbourne
School of Computing and Information Systems SWEN90007
Software Design and Architecture. Semester 2, 2019

Project Part 4: Reflection and Report

Online Mobile Phone Store Management System

Student ID: 671499,
686141

Student Name: Yixiong Ding,
Ruifeng Luo

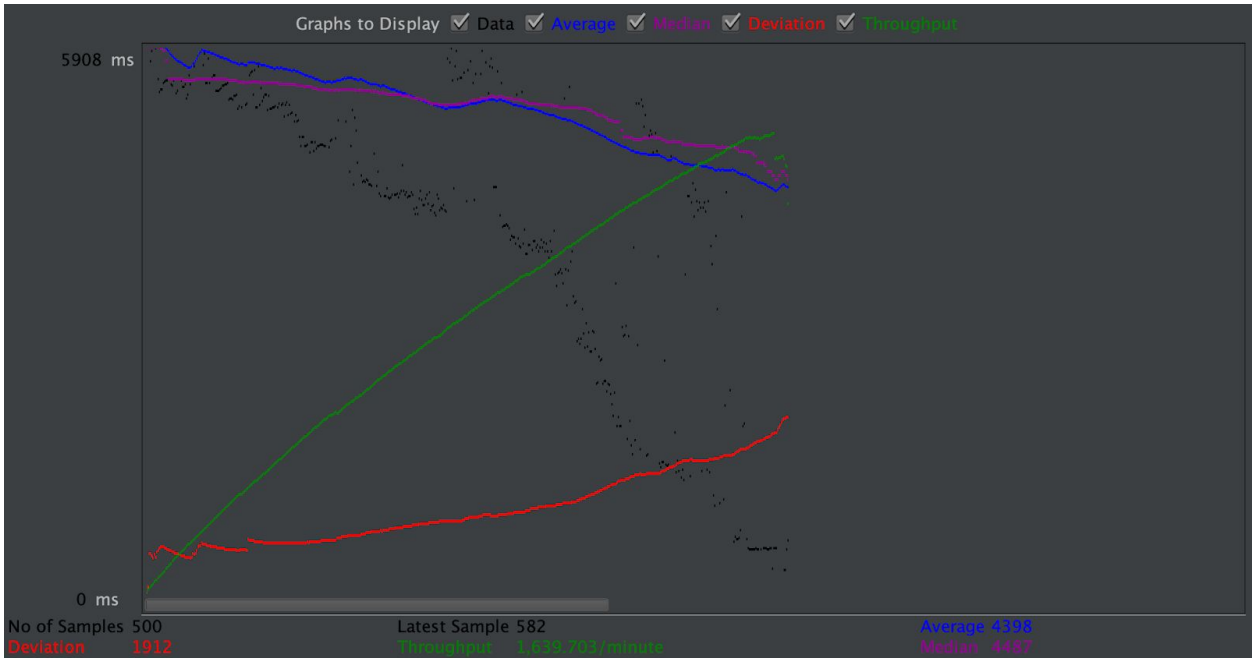
1. JMeter performance test

Firstly, we use Apache Jmeter to evaluate the overview performance of our system. We have implemented three different experiments to test the system performance. It is based on three different functionalities below.

User Login:

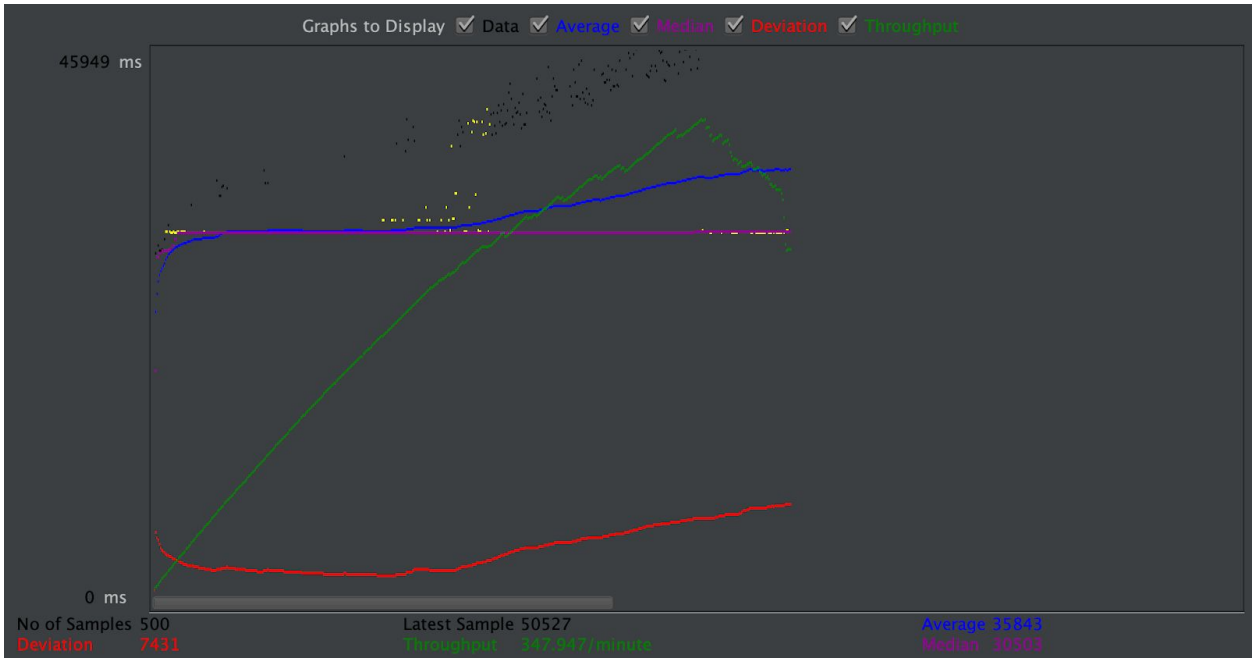
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
login_customer	250	4328	621	11436	2422.99	0.00%	21.3/sec	41.79	5.36	2013.0
login_admin	250	3537	507	9498	1658.36	0.00%	21.4/sec	40.49	6.96	1938.0
TOTAL	500	3932	507	11436	2113.50	0.00%	40.6/sec	78.37	11.72	1975.5

View Orders:



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
View orders	500	4398	252	11761	1912.16	0.00%	27.3/sec	122.36	6.37	4585.0
TOTAL	500	4398	252	11761	1912.16	0.00%	27.3/sec	122.36	6.37	4585.0

Register customers:



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
create customers	500	35843	18760	60629	7431.71	61.20%	5.8/sec	6.85	2.93	1209.1
TOTAL	500	35843	18760	60629	7431.71	61.20%	5.8/sec	6.85	2.93	1209.1

According to the results of the experiments above, we find that the results of “user login” performed better than “view orders”, which then performed better than “Register customers”. From these experiments, we can see that our system has poor performance on the throughput and average response time.

Discussion

We have implemented and applied pattern in our system, these patterns may either decrease or increase performance. In this section, I will have discussions on these patterns’ performance.

Identity Map

In our system, the identity map pattern is to maintain a mapping between objects in the database and references to the instance of that object. Every object in the database is assigned a unique ID. It increases our system performance by reducing the number of calls to the database. Based on the caching principle, it stores data in a faster-access location to decrease the time it takes to access the data and eliminate network latency.

Unit of Work

The unit of work pattern in our system keeps track of which domain objects have changed or new objects have created, and update those changes in the database. Furthermore, it maintains a list of objects affected by a business transaction and coordinates the writing out of changes. It saves multiple connections made for each change, which is expensive. And also, it eliminates network latency and optimize throughput performance.

Lazy Load

The lazy load pattern is to reduce the amount of data read from the database in our system. Furthermore, lazy loader loads dummy data which contains null data. Lazy loader only loads data from the database when we try to access that data. If an object is never accessed, the object remains with dummy data in the application and never reads the corresponding data from the database. The design pattern significantly improves performance when only portions of data are required from the database. It avoids unnecessary memory and connection to the database so that it decreases latency in our system.

DTO and Remote Facade

The combination of both data transfer object and remote facade pattern avoid the issue of returning multiple values and eliminating any design-time type checking. The data transfer object pattern assemble attributes into a single object, and the single object is then serialized and sent over the network. The combined patterns decrease latency and increase the number of works that can be done in a given amount of time.

Pessimistic Offline Lock

The pessimistic offline lock is to prevent conflicts between concurrent access to data in the database by locking the required data over the entire logic transaction. In our system, it decreases the performance of our system since it only allows only one business transaction at a time to the data. The design pattern increases the latency of the system.

Principles

In terms of performance in the project, we adopted the Bell's principle, which indicates simple designs often show better performance than more complicated designs. By adapting three-tier architecture, we make simple and loose coupling in our system design. Our system is divided into the presentation layer, domain layer, and data source layer. The presentation layer is focusing on the user interface, which takes requests from the user and provides a user with responses. The domain layer is to implement the domain logic and interact with the data source layer and presentation layer. The data source layer is to perform CRUD operations on the database. In the three-tier architecture design, each layer is independent with each other and reduce the complexity of the system. It indicates Bell's simple design principle and having better performance on the system.

In the pipelining principle, the combination of data transfer object and remote facade design pattern assemble attributes into a single object, and the single object is then serialized and sent over the network. Instead of several method calls to obtain all attributes' information, the remote facade packages these into a single call. By using the pipelining principle, it would increase the throughput and decreasing the time that users wait for access to resources.

In the caching principle, it indicates storing data in a faster-access location to decrease the time it takes to access the data. Every time data has been read from a slower disk to memory, these data would be kept in the memory for a while. In our design, the identity map pattern reads data from the database (slow storage medium) to the main memory whenever it requires to access the database. Otherwise, it returns an instance from the identity map.

Reflection

In the feature B part, we added a new ‘customer’ role and patterns, such as session distribution, concurrency, and security. These patterns have an impact on the original architectural design. Furthermore, we eliminated some original designs or patterns that contain old design and new items. For example, as we added a new role ‘customer’ in feature B, we changed the schema design in our database. We implemented a single inheritance design, and we designed both roles ‘customer’ and ‘admin’ to take inheritance from ‘user’.

As to security in our system, we implemented features about authentication and authorization using Apache Shiro library. We eliminated our authentication design in feature A using web.xml (embed username and password into web.xml file). And also, we implemented sessions and cookies by using the Shiro library.

For concurrency, we used the implicit lock and pessimistic offline lock whenever accessing the database and performing CRUD operations. To achieve concurrency, we added a new table to store the locking status of data to conflicts between concurrent business transactions. The impacts of the implementation would slow down the system performance since it allows only one business transaction at a time to access data.

As to maintainability in our system, the following aspects will be discussed below,

- **Single table inheritance** - we implemented the Single table inheritance pattern in database table design, which allows us to extend more different kinds of users in the future, and no need to change the original table design.

- **Data mapper** - since data mapper has better performance on loose-decoupling and compatibility, we could easily extend our domain logic, and no need to change the underlying data when the domain logic gets more and more complex.

- **Pessimistic offline lock and implicit lock** - we adopted both locks for improving our system concurrency performance to avoid data conflicts. As we built the frame for the concurrency extension, there is no need to change the system design for accessing data when the degree of concurrency gets higher.

- **Authentication and authorization** - we implemented authentication and authorization in our system using Shiro lib which is scalable. we can easily extend those fine-grained access restrictions such as links, buttons or data displayed based on the user permissions. In this case, there are no many changes needed to make and add the authorization types.