Jurnal Elektronik Ilmu Komputer Udayana
Volume 12, No 3. Februari 2024

p-ISSN: 2301-5373
e-ISSN: 2654-5101

Database Performance Optimization using Lazy Loading with Redis on Online Marketplace Website

Albertus Ivan Suryawan[a1], Agus Muliantara[a2]

[a]Informatics Department, Udayana University
Bali, Indonesia
[1]albertusivan15@gmail.com
[2]muliantara@unud.ac.id (Corresponding author)

Abstract

With the pandemic lasting over 2 years, many businesses start to adapting a digital approach of their business to stay alive. While in the same time, a number of users of digital platform also skyrocketed due to physical contact restriction policy. This causes a performance hit toward several online services such as an online marketplace due to high network traffic from many users accessing it in the same time, including latency issues. In this research, the authors try to implement an application-level caching with an in-memory database, Redis, using Lazy Loading approach. Beside implementing caching, authors also compare the performance of using cache and not using cache by load testing both implementation using similarly built application. Based on the result, there is a performance gain of 38-65% based on the load and scenario by using application-level caching.

Keywords: *Cache, Cache-Aside Pattern, Lazy Loading, performance optimization, application-level caching, in-memory database*

1. Introduction

With the pandemic lasting over 2 years since March 2020, many businesses start to migrating their operation to digital platform, in order to survive due to physical contact restriction policy. Some start to use social media as an e-commerce platform, while others making their own online services from which accessible through online. The latter is the case of a certain business called Business X, which name cannot be disclosed, in which they start building their own online marketplace business to continue selling several products. Over a year after starting an online marketplace, they start noticed a high latency issue while navigating through the website, due to high network traffic. The authors try to analyze the infrastructure used, and found out there is a bottleneck on the database due to high traffic mention earlier. To address this issue, the authors try to implement caching into the application, in form of application-level caching.

Caching is often being used to reduce load coming into the main memory, which commonly known to be slow, by storing data that frequently accessed in the cache itself. Application-level is a type of caching in which the implementation of the cache is done by the developer itself based on the condition. Usually, this type of cache is implemented in the server-side of the application, and mostly relied on in-memory database such as Redis[1], [2]. In order to avoid additional strain on in-memory database due to excessive data stored inside, Cache-aside pattern or Lazy-Loading, will be used to limit when data should be stored. Cache-aside pattern is a type of data synchronization where data is being stored in the cache on demand[3].Consequently, instead of storing all data directly on the cache, only previously requested data will be stored inside until its expired or an update for the that data occurred in main database.

There are several research that has been done regarding caching such as on research done by Saldhi et al. [1] which analyze performance difference between two Cache System namely, InfiniSpan and Hazelcast. In that research, the author suggests to do a comparative performance analysis across different cache pattern. Another research with similar interest that has been done is [2] in which a performance between MySQL database were compared with Redis as cache database. Based on the result, there are some degree of performance gains by using Write-Through pattern that the writer has implemented. Although [2]may be similar to this research, there is a difference in the type of data

synchronization pattern being used, where author used Cache-Aside pattern instead of Write-Through pattern.

2. Reseach Methods

Due to the nature of the subject, the methodology used on this research mainly consist of 5 stages as shown on Figure 1, including system requirement analysis, database modeling, cache strategy modeling, application development, and performance analysis.

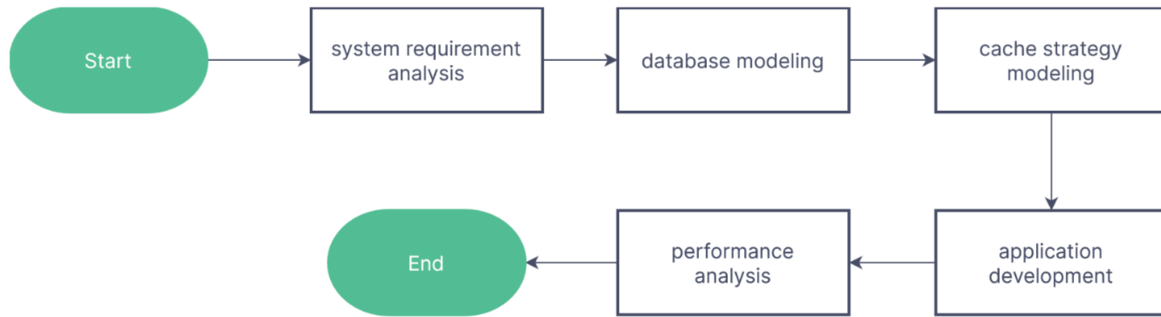

Figure 1. Research Methodology

2.1. System Requirement Analysis

System Requirement Analysis is the first stage of System Development Life Cycle (SDLC) where the developer will conduct an analysis to list any requirement needed for the system. On this research, there are 3 main feature that needed to be existed including, user registration and login; get list of available products and get product's detail; and checkout an order.

2.2. Database Modeling

In this research, there are 2 types of databases being used for the application, a relational database using MySQL, and an in-memory database namely Redis. Redis is chosen due to its popularity as a cache management system with high flexibility of usage such as shown in [4]. MySQL will be used as the main database, where all of the data will be stored, and Redis will be used as a cache layer, where all cached data will be store based on the cache strategy modeling being used. MySQL database consist of 4 tables as shown in **Figure 2** and **Figure 3**.

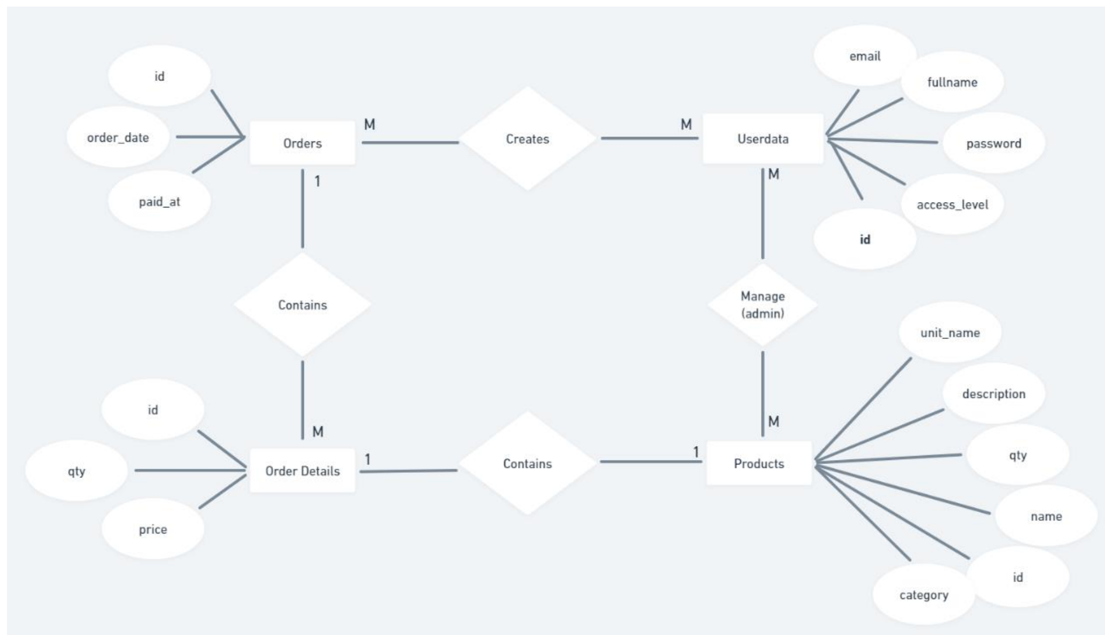

Figure 2. Entity Relational Diagram

 p-ISSN: 2301-5373
e-ISSN: 2654-5101

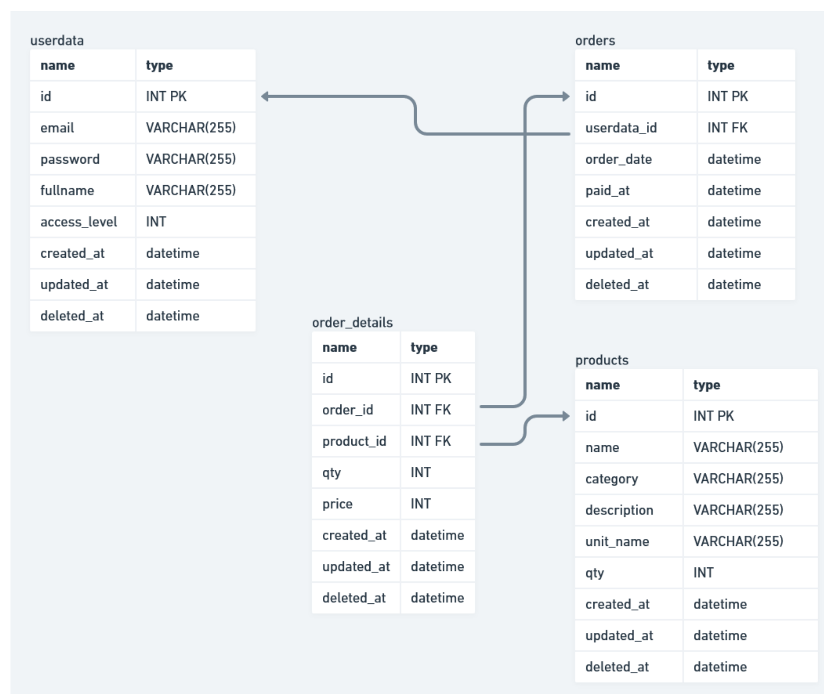

Figure 3. Relational Database Model

Table *userdata* contains all user data including email, password, full name, and access privilege level. Table *orders* and *order_details* contain all order data such as product bought, quantity, price at the time bought, and user who ordered it. Table *products* contains all product data such as product name, price, and remaining stock. While MySQL can be model after their table, Redis in the other hand is a key-value database in which described as shown in Table 1. Standardized naming scheme for Redis' key is being implemented to keep the caching logic simple and avoid several problems in the future such as name collisions [5]

Table 1. Redis Naming Scheme

Field	Naming Scheme	Expiry Duration
Userdata Cache	user:<userdata_id>	1 Hour
Session Cache	session:<session_id>	5 Minutes
User-Session Index Key	user-session:<user_id>	5 Minutes
Product Detail Cache	product:<product_id>	1 Hour
All Product Summary	product:summary	1 Hour

2.3. Cache Strategy Modeling

There are several ways to implement data synchronization between main database and the cache. As mention in the introduction, the author chooses to implemented a Cache-Aside pattern in which data is stored in the cache when there is a request for it, and retain in the cache until its expiry time reached or the corresponding data is updated in the database. In **Figure 4**, the overall flow of retrieving data using Cache-Aside pattern is shown.

First, application will try to fetch the data from the cache, if the data existed, it immediately returned. Otherwise, application will try to fetch the requested data from the main database, and store its value to the cache if data found in the database, and then its value returned. In **Figure 5**, the overall flow of updating data is shown, where data will be discarded from the cache after data successfully updated in the main database. In this application, deleted data is a data in the main database that has field *deleted_at* set to non-*NULL* value (i.e., *timestamp* of data being deleted).

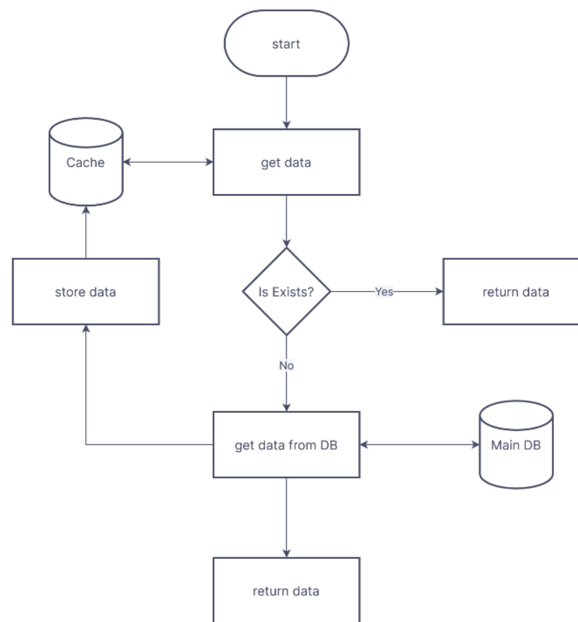

Figure 4. Cache-Aside Data Retrieval

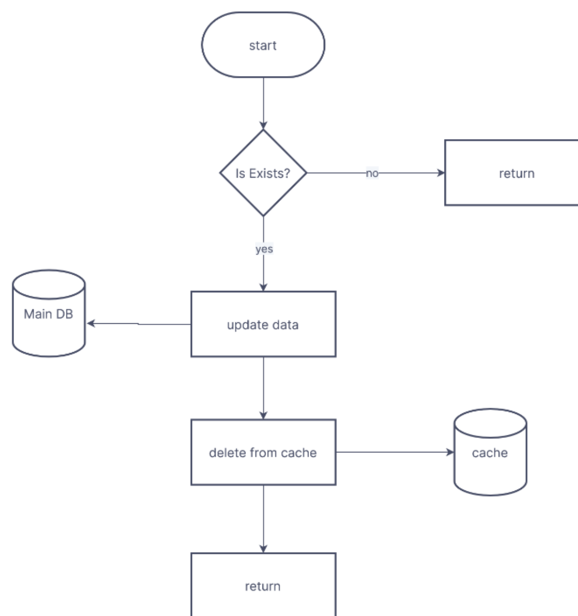

Figure 5. Data Update Flowchart

2.4. Application Development

In this stage, application is developed in Go Language. This language is chosen because it's known for the performance especially as a server-side application. There are 3 main components needed, *controller, service,* and *repository*. Controller is part of application in which incoming requests processed before passed to *service* to be consumed. Service is part of application in which the business logic lies. In *service,* program will also call *repository* to fetch data whether it's from cache or main database. Repository is part of application which process all database-related operation, whether from main database or from cache layer. In *repository,* all database connections are handled by each officially supported connector driver for Go Language.

3. Result and Discussion

In this research, the performance of the proposed application is measured by running a load test between proposed application and a similarly built application without cache being used. Load test is

 p-ISSN: 2301-5373
e-ISSN: 2654-5101

done using k6 with fixed time of 300 seconds and virtual users, which denoted a concurrent connection to the application in a time, of {100, 200, 500}. This load test measures the time from sending requests to first byte received for 3 types of requests scenario. The overall result is shown in **Figure 6**

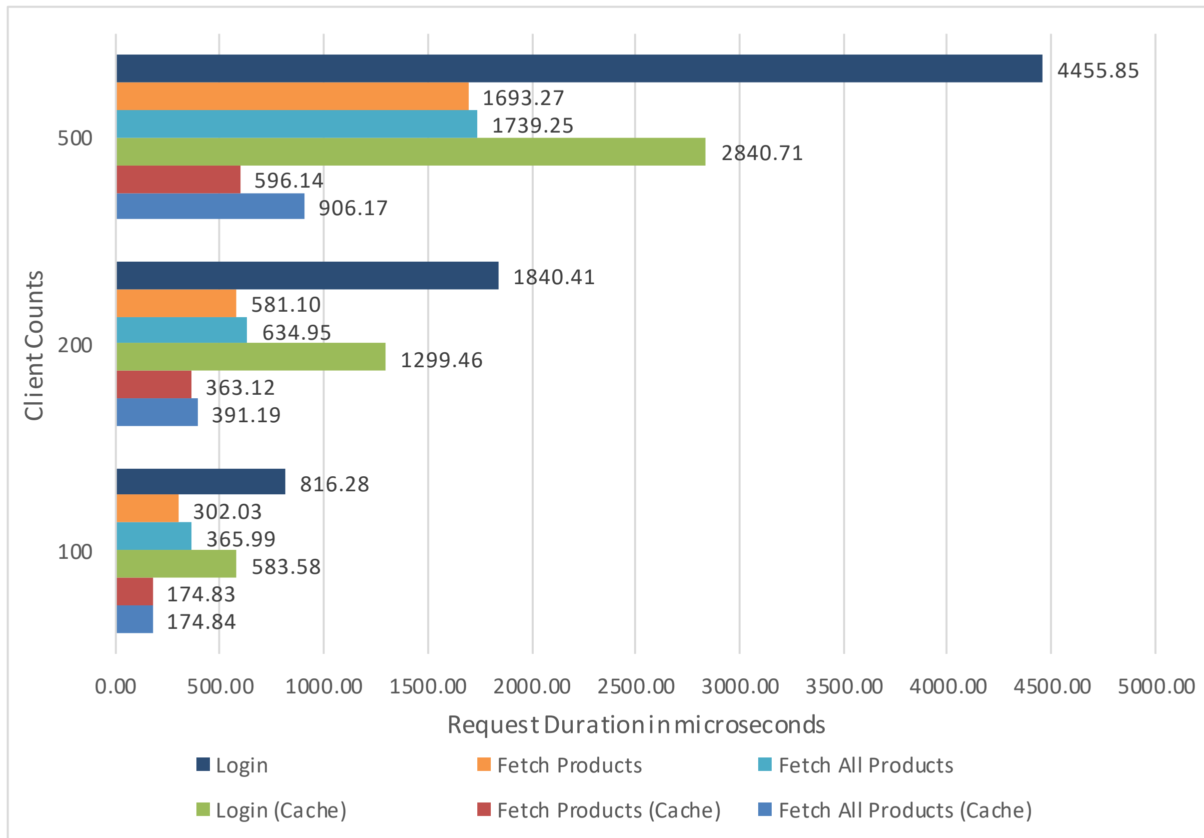

Figure 6. Overall performance of each scenario in term of request duration

3.1. Login Scenario

This scenario simulates users' action of login with their account which simulate cold data, data that has not been stored in the cache, being fetch and stored in the cache for future use (i.e., re-login to get new *sessionID* for API authentication). Result of this interaction is shown in Table 2. Based on the result, by implementing cache on login endpoint, results in performance gain about 28-36% depending on the load in average. There are some performance degradations as shown by the minimum value of using cache when interacting with 500 concurrent users of about -343% relative to using no cache.

Client Count	Request Duration (*ms*)					
	Avg	Min	Max	Avg	Min	Max
	Cache			No Cache		
100	583.58	1.68	3633.21	816.28	3.20	4859.88
200	1299.46	2.10	6024.22	1840.41	7.22	9693.84
500	2840.71	25.43	16179.62	4455.85	6.00	22780.09

Table 2. Login Scenario Load Test Result

3.2. Fetch All Products Scenario

This scenario simulates users' action of browsing through the products catalogue which simulate repetition of fetching the same data over and over. Result of this interaction is shown in Table 3. Based on the result, by implementing cache on fetch all product endpoint, results in performance gain about 38-53% depending on the load in average.

Client Count	Request Duration (*ms*)					
	Avg	Min	Max	Avg	Min	Max
	Cache			No Cache		
100	174.84	0.51	2767.48	365.99	1.04	1837.42
200	391.19	1.00	4810.10	634.95	0.77	6466.26
500	906.17	1.16	9499.17	1739.25	1.00	12837.18

Table 3. Fetch All Products Scenario Load Test Result

3.3. Fetch Product's Detail Scenario

This scenario simulates users' action of browsing details for product they have interest with, which simulates random repetition of fetching the same data. Result of this interaction is shown in Table 4. Based on the result, by implementing cache on fetch product's detail endpoint, results in performance gain about 38-65% depending on the load in average.

Client Count	Request Duration (*ms*)					
	Avg	Min	Max	Avg	Min	Max
	Cache			No Cache		
100	174.83	0.52	1976.53	302.03	0.54	1810.73
200	363.12	0.53	2762.30	581.10	0.63	7013.00
500	596.14	1.12	12169.89	1693.27	1.00	13770.91

Table 4. Fetch Product's Detail Load Test Result

4. Conclusion

There are many ways to improve performance of a web-based application such as an online marketplace, including implementing a cache layer. Based on the result shown in the previous section, the implementation of cache in the application can be considered as a successful attempt. Implementation of cache do increase the application request performance by 38-65% based on the scenario, with repetition of fetching the same data (i.e., all products summary, and product's detail) gained at least 38% on high load. Besides performance gains, the authors also found out that number of concurrent users also affect the performance of the application as shown in the results, whereas the client count increases, the duration it takes to complete the request also increases. For future work, authors suggest make another approach of cache implementation.

References

[1] H. Salhi, F. Odeh, R. Nasser, and A. Taweel, "Benchmarking and Performance Analysis for Distributed Cache Systems: A Comparative Case Study," 2018, pp. 147–163. doi: 10.1007/978-3-319-72401-0_11.

[2] M. I. Zulfa, A. Fadli, and A. W. Wardhana, "Application caching strategy based on in-memory using Redis server to accelerate relational data access," *Jurnal Teknologi dan Sistem Komputer*, vol. 8, no. 2, pp. 157–163, Apr. 2020, doi: 10.14710/jtsiskom.8.2.2020.157-163.

[3] Microsoft, "Cache-Aside pattern." https://learn.microsoft.com/en-us/azure/architecture/patterns/cache-aside (accessed Oct. 01, 2022).

[4] R. K. Singh and H. K. Verma, "Effective Parallel Processing Social Media Analytics Framework," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2860–2870, Jun. 2022, doi: 10.1016/j.jksuci.2020.04.019.

[5] J. Mertz and I. Nunes, "A Qualitative Study of Application-level Caching," Oct. 2020, doi: 10.1109/TSE.2016.2633992.