Distributed Producer Consumer Model

Ali Zamin[1][0000-1111-2222-3333]

[1] University of Memphis, Memphis TN 38152, USA
zaminali@memphis.edu

Keywords: Distributed System, Producer-consumer problem, Message-Queue system, Processor, Credit Card, Java.

1 Abstract

In this project, I present the architecture and implementation of a credit card processing system using a distributed producer-consumer model. The system involves multiple producers responsible for submitting credit card payments, a messaging system for handling communication, and processors for processing the submitted credit card transactions. I utilized a simplified message queuing system to enable communication between producers and processors in the distributed system. This report provides a comprehensive overview of the project, including the system design, implementation details, evaluation, and scalability analysis.

2 Introduction

In today's digital era, the rise of digital transactions has revolutionized the way we conduct our financial operations. Among these, credit card transactions stand at the forefront, enabling seamless and instantaneous payments across the globe. As the volume of credit card transactions continues to surge, the demand for efficient, secure, and scalable credit card processing systems becomes paramount. The task of processing these transactions in a timely manner while ensuring data integrity and reliability poses significant challenges, necessitating the development of advanced solutions.

This project delves into the realm of distributed systems to address the complex task of credit card transaction processing. At its core, the project revolves around the concept of a distributed system, which refers to a network of interconnected nodes that collaborate and communicate to achieve a common goal. Unlike traditional monolithic systems, distributed systems offer several advantages, including improved fault tolerance, scalability, resource availability and transparency.

To tackle the challenge of credit card transaction processing, we adopt the producer-consumer model. This model is a fundamental paradigm in distributed computing, where data production and consumption are decoupled to enhance efficiency and parallel processing. The design of this system is employed in a way that a producer and consumer are not directly linked to each other but through a shared memory buffer. This helps the efficiency of the system as the producers and consumers can directly access the data without being connected and dependent on each other. In our context, multiple components, acting as producers, are responsible for generating credit card transactions. These transactions are then channeled to a centralized message queue system, acting as the consumer, which orchestrates the processing and distribution of transactions to specialized processors.

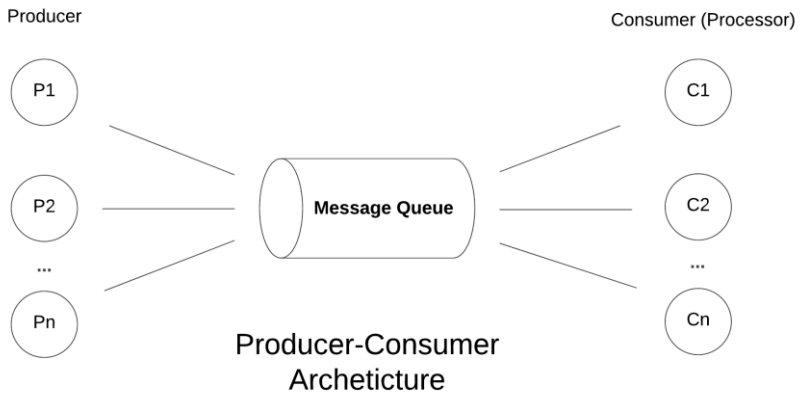
As part of this project, we build a simple yet powerful in-memory message queue system to facilitate communication between producers and processors. This messaging infrastructure serves as the backbone of the distributed credit card processing system, ensuring reliable and efficient transaction handling. By adopting such an architecture, we attempt to demonstrate a deeper understanding of distributed communication mechanisms and the organization of data flow in distributed systems.

3 Related Work

Although, I couldn't find any journals or papers related to credit card transaction processing by distributed systems, there are couple of research papers in ACM digital library which go over producers-consumer architecture, specifically for presenting better communication system solutions. One of the most related papers found was "A Scalable Multi-Producer Multi-Consumer Wait-Free Ring Buffer" by Andrew Barrington, Steven Feldman, and Dr. Damian Dechev. In this study, they implemented a communication system for producer-consumer model, called "ring buffer" or "cyclical queue" which is a First In, First Out (FIFO) queue that stores elements on a fixed-length array. They claim it build the first array-based wait-free ring buffer.

Based on the topics we studied in this class for distributed communication systems, the most suitable system I found to be appropriate for this project was the message-oriented system, called message queue system, as it was something I was familiar with and it was not seemed to be used in other ACM papers, in the context of producer-consumer model.

4 Design



The design of our credit card transaction processing system follows a distributed producer-consumer model with a manual in-memory message queue, as shown in the figure above. I have developed three key components: producers, message servers, and processors. Producers are responsible for submitting credit card payments, and they communicate with the message queue to enqueue transactions. Message servers, acting as intermediaries, facilitate communication between producers and processors. Processors, on the other hand, dequeue transactions from the message queue and perform the required credit card processing tasks.

4.1 Tools

To build this system program, Java programming language was used, using Eclipse IDE. Java has lots of helpful libraries and packages for building distributed systems, as such, I used one of the library, LinkedBlockingQueue, for handling the messaging queue.

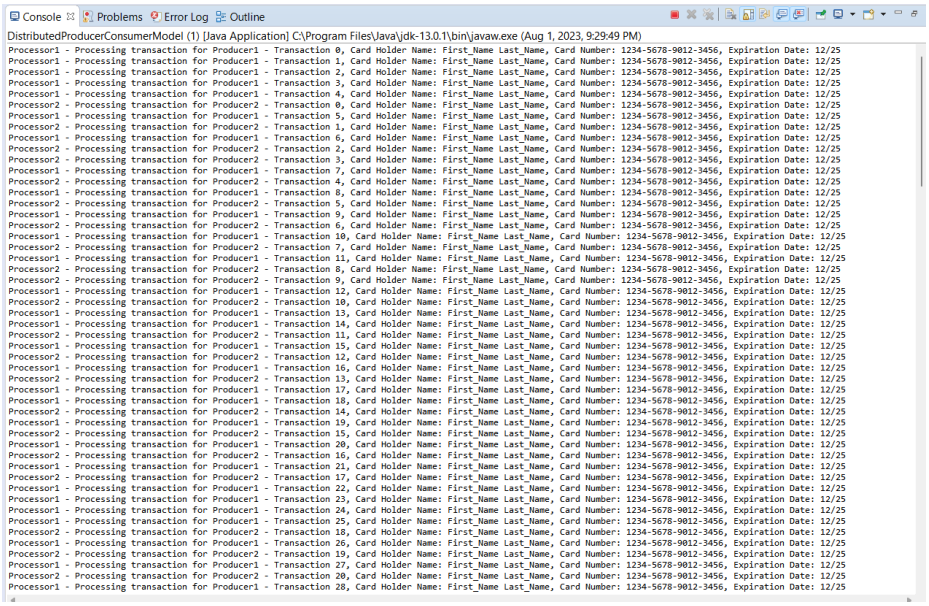
LinkedBlockingQueue is an implementation of the BlockingQueue interface provided in Java. It is a bounded queue backed by a linked list. In our context, it's used to store credit card transactions in the message queue. If the queue is empty, the dequeue method will block (wait) until a new transaction is available to retrieve. Similarly, if the queue is full when a producer tries to enqueue a transaction, the enqueue method will block (wait) until there is space available in the queue.

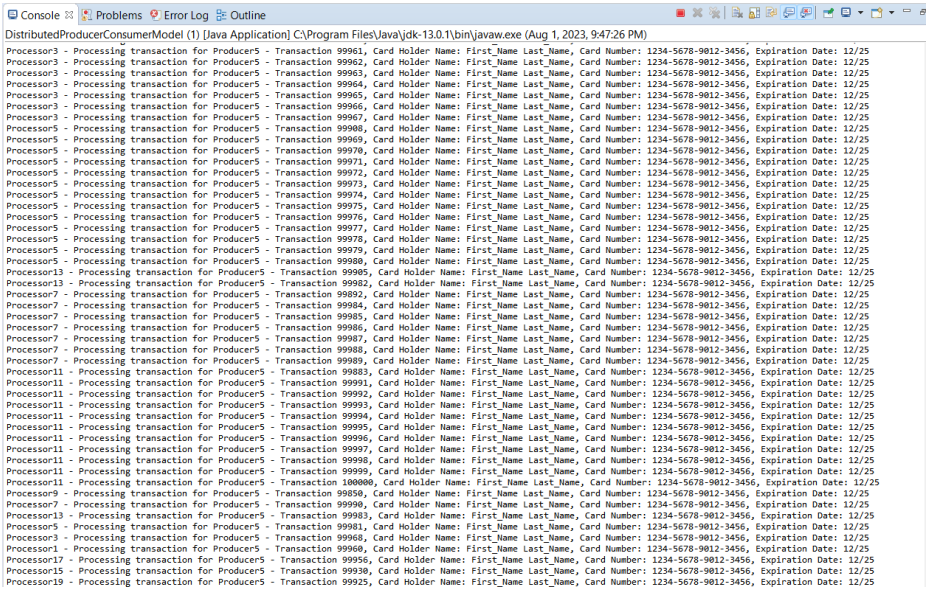
We structured the program using Java's object-oriented design, where we created separate classes for producer, processor, message, credit card transaction, and the main producer-consumer program. This design helps in creating multiple producers, processors, and shared memory (message queue) for distributing the load.

The two most important methods/functions in our program are the enqueue and dequeue methods, which helps in sending and receiving the data in queue. The enqueue method is used by the producer(s) to send data to the queue and dequeue method used by processors to retrieve data from the queue.

5 Analysis

For analyzing the program's scalability and efficiency, I tested and ran the program with different number of producers and processors threads for 100,000 transactions in each iteration. When running the program with 2 producers, 2 message queues, and 2 processors, the program took about 10.82 seconds. This amount of time seems normal for 100,000 data points. On the second iteration, I increased the number of both producers and processors from 2 to 5. This time the transaction execution time took 24 seconds, which increased the time by 14seconds (129%). On the third iteration, increasing the number of producers and processors from 2 to 20 took 1 minute and 44 seconds of time, which is a significant increase as it increased the time by 1 minute and 33.18 seconds (861%).





Although, deciding the threshold for performance and scalability of the system based on time consumption here can be challenging, as it's subjective and depends on the context. However, if we were to judge by time, based on the results obtained, it is observed that there are several factors that can influence the performance and scalability. One of the critical performance factors is the number of producers and processors employed in the system. The increase in the number of producers and processors, increases the transaction time significantly, which directly affects the scalability. In the distributed system, a system is considered scalable when its performance is unaffected by the addition of more nodes, computers, or processors to the system. If we are to tie the performance with the amount of time it takes for transactions, then this architecture doesn't seem to scalable. Though this program is a simple implementation of producer-consumer architecture, imagine if we have thousands of computers connected to this system with millions of credit card transactions daily? (which is not uncommon). This system would really be inefficient and unscalable. That being said, the real scalability of the system can be assessed when testing in the real-world practical environment.

6 Conclusion

In this project, I was able implement a distributed producer-consumer model for credit card transaction processing, using message queue system and other concepts learned in this class. While given the time frame, building a simple system was prioritized, this project has provided valuable practical experience in building distributed systems

and lays the groundwork for further advancements to create a robust and scalable solution for processing real-world credit card transactions.

7 Reference

Barrington, Andrew, et al. “A Scalable Multi-producer Multi-consumer Wait-free Ring Buffer.” *ACM*, Apr. 2015, https://doi.org/10.1145/2695664.2695924.

Feldman, Steven, and Damian Dechev. “A Wait-free Multi-producer Multi-consumer Ring Buffer.” *Applied Computing Review*, vol. 15, no. 3, Association for Computing Machinery, Oct. 2015, pp. 59–71. https://doi.org/10.1145/2835260.2835264.

Moiseenko, Ilya, et al. “Consumer / Producer Communication With Application Level Framing in Named Data Networking.” *ACM*, Sept. 2015, https://doi.org/10.1145/2810156.2810160.

Luo, Ying, and Xianping Wang. “The Study of the Classic Producer-consumer Problem in a Series of IT Courses.” *ACM*, Oct. 2020, https://doi.org/10.1145/3368308.3415412.

GeeksforGeeks. “Producer Consumer Solution Using Semaphores in Java Set 2.” *GeeksforGeeks*, Nov. 2019, www.geeksforgeeks.org/producer-consumer-solution-using-semaphores-java.