Beijing-Dublin International College

COMP3008J Distributed Systems

Feasibility Analysis of Distributed System Migration: Optimizing the Retail Operations of X Company

Author

Le Liu(22207256)

Date

December 11, 2024

Contents

1 Company Business Background

X Company is a large retail enterprise with operations spanning multiple cities, numerous retail stores, and offices. The company has an independent IT department. Currently, the business is divided into two primary segments: offline retail stores and an online e-commerce platform(Like Walmart). With rapid business growth, frequent inventory turnover, increasing order volumes, and expanding staff size, the company faces rising operational complexity.

To support core business operations, X Company currently employs a centralized system architecture. Business logic and data storage are deployed in a single data center, and all client requests (including retail store systems, online platforms, and office systems) are processed through a central server. While this architecture initially offered low cost, ease of management, and rapid deployment, it is now revealing several critical limitations as the business grows:

1. **High latency across regions**: The distribution of retail stores and the centralized data center across different locations leads to long response times due to distant data transmission, negatively affecting user experience.
2. **Single point of failure risk**: A centralized system means that any failure in the data center could disrupt company-wide operations, including store transactions, online order processing, and internal management.
3. **Low scalability**: Hardware limitations make it challenging to handle surges in traffic during peak business periods (e.g., promotional events).
4. **Difficulty in online-offline integration**: The current architecture struggles to synchronize inventory and order data between online platforms and offline stores in real-time, impacting operational efficiency and customer experience.

Given these challenges, the existing centralized architecture can no longer meet the company's demands. To support future business expansion and improve system performance and reliability, X Company urgently needs to migrate to a distributed system to build a highly efficient, reliable, and scalable IT infrastructure.

2 Microservices Architecture Design

X Company needs to migrate to a **microservices architecture**. By modularizing business functions and deploying them independently, this architecture can significantly improve scalability, availability, and security, while enabling seamless online-offline integration.

Microservices Segmentation

Based on X Company's business characteristics, the system is divided into the following independent microservices, each performing a single function and deployed independently:

Inventory Management Service

- **Responsibilities**: Track all inventory levels, update stock, handle item check-in and check-out, and synchronize with the order system and procurement system.
- **Technical Implementation**: Following Amazon's approach, **DynamoDB** is used to store high-availability inventory data, and **SQS (Simple Queue Service)** is employed for asynchronous task processing. Whenever an order is created, the system notifies the inventory service via a message queue to decrement stock and update other systems.

Order Management Service

- **Responsibilities**: Handle user orders from online and offline channels, track order status, process payments, and manage order history.
- **Technical Implementation**: Utilize **Apache Kafka** to build an event-driven architecture. When an order is created, Kafka publishes the event to a message queue, and other services (e.g., payment service, inventory service) consume the event and perform the necessary operations. This design decouples services, improving scalability and fault tolerance.

User Account Management Service

- **Responsibilities**: Manage user registration, login, and account data storage, and integrate with the payment service.
- **Technical Implementation**: Use the **OAuth 2.0** protocol for user authentication and authorization. **JWT (JSON Web Tokens)** ensures secure and efficient user identity transmission across services, enabling seamless cross-service authentication.

Payment Service

- **Responsibilities**: Process user payment requests, integrate with third-party payment systems, and synchronize with the order management service.
- **Technical Implementation**: Integrate with third-party payment services (e.g., PayPal, Stripe) via APIs to simplify payment processing. These services support multiple payment methods (credit cards, installment payments, digital wallets), enabling quick deployment and ensuring secure transactions. There is a need to pay a fee to a third party payment provider, as well as becoming payment dependent, but I still believe that choosing to use their services directly, rather than building your own payment system, is a better option at this stage of business development.

Technical Features of Microservices

The microservices architecture is designed to address X Company's requirements for scalability, availability, and security:

Scalability

- **Horizontal Scaling**: Add more microservice instances to handle peak traffic demands (e.g., Black Friday, holiday promotions).
- **Independent Scaling**: Scale each microservice independently based on actual traffic demand, avoiding resource waste.
- **Load Balancing**: Use load balancers (e.g., AWS ELB, Nginx) to distribute user requests across microservice instances, optimizing resource utilization.

Availability

- **Service Redundancy**: Deploy multiple instances of each microservice across different availability zones to prevent single points of failure.
- **Failover Mechanism**: Automatically switch to backup instances in case of service failure, ensuring business continuity.
- **Data Backup and Recovery**: Regularly back up critical data (e.g., orders, inventory) and implement fast recovery mechanisms to avoid catastrophic data loss.

Security

- **Data Encryption**: Use **AES** technology to encrypt stored data, protecting user personal information and company-sensitive data.
- **Network Security**: Employ **TLS/SSL** encryption for user interaction data to prevent man-in-the-middle attacks.
- **Web Application Firewall (WAF)**: Use WAF to prevent common attacks such as SQL injection and cross-site scripting (XSS).
- **Key Management**: Adopt Amazon's practice by using **KMS (Key Management Service)** to encrypt stored keys and ensure only authorized services can decrypt them.

3 Distributed File System

Using the Ceph architecture, Company X handles massive data storage objects, including product catalogs, orders, video advertisements, and transaction records. Different types of data require varying storage solutions, making a multi-type storage-compatible file system essential.

Ceph supports encrypted storage and allows for categorized management of access permissions across different levels of storage objects. For file access, Ceph provides multiple access structures to meet diverse application needs. Regarding cache updates, it supports both strong consistency and eventual consistency, allowing for flexible switching based on the specific use case.

Ceph also features automated data backup management, significantly improving reliability. However, Ceph exhibits weaker support for small files, as it allocates a fixed storage block size for files, leading to potential storage inefficiency when handling small files.

This design reflects the distributed systems paradigm, prioritizing scalability, fault tolerance, and flexibility in managing complex and large-scale data storage needs.

4 Communication Protocols

In a microservice architecture, efficient communication protocols are essential for ensuring system performance and reliability. Company X's distributed system must support high performance, high concurrency, and high stability, as well as flexibility and security, to guarantee smooth communication between components. For instance, when a customer places an order, the order system must communicate with the inventory management system in real time to ascertain the inventory status. Similarly, after the customer makes a successful payment, the payment service must notify the order management system in a timely manner to update the order status. These scenarios illustrate the necessity for robust communication protocols.

Based on our analysis of Company X's specific requirements, we propose a communication protocol combination of RESTful API + gRPC. This solution allows us to address the client-server interaction needs and efficient communication between the back-end microservices. While it may increase deployment complexity and maintenance costs, we believe that, given the company's existing IT capabilities, this combination will significantly enhance user experience and system performance.

RESTful APIs are well-suited to the following scenarios:

The interaction between a client and a back-end service is an example of this. A user may submit an order or query the status of an order, for instance. Communication between services and external systems, for example, initiating a transaction with a third-party payment service. RESTful APIs have become the industry standard for client-to-server communication due to their broad compatibility and ease of use.

gRPC is the optimal choice for communication between back-end microservices, excelling in the following scenarios:

The need for high throughput and low latency requirements is a key factor. An example of this would be the real-time data synchronisation between an inventory service and an order service.

This combination of protocols strikes a balance between performance and flexibility.

5 Final Thoughts

Through the migration to a distributed system, Company X can effectively address the growing complexity and challenges in both its online and offline retail operations. The report highlights the distributed system architecture as follows:

- **Microservices Architecture**: The system is decomposed into microservice modules based on busi-

ness requirements.

- **Distributed File System**: DynamoDB is utilized for business data storage, while Ceph is employed for unstructured data storage.
- **Communication Protocols**: RESTful API and gRPC are adopted as the main communication protocols between services. Additionally, Kafka and Simple Queue Service (SQS) are introduced for asynchronous task handling.

With the continuous development of cloud computing, integrating distributed systems with cloud-native technologies has become an increasingly popular choice for large enterprises. As business complexity grows, the need for more intricate distributed systems arises, requiring thorough research and rigorous architectural design by IT departments.

Of course, this demands constant adjustments—there is no one-size-fits-all solution, only the ongoing exploration of possibilities through practical experience. Company X can leverage network technologies to accommodate its business growth and even drive its retail operations. This hinges on the company's emphasis on networking, strengthening its IT department, allocating more resources, and choosing the appropriate technologies—all of which are worthwhile investments.