INTE 22253 - Distributed Systems and Cloud Computing

Mini Project Report

Group 01

Department of Industrial Management

Faculty of Science

University of Kelaniya

Sri Lanka

June 2023

Declaration

Declaration by the Students

I hereby certify that no part of this assignment has been copied from any other student works or from any other source except where due acknowledgement is made in the assignment. No part of the assignment has been written/produced for me by any other person except where such collaboration has been authorized by the subject lecturer/tutor concerned. I hold a copy of this assignment that I can produce if the original is lost or damaged.

1. IM/2019/031 - Fathima Riztha
2. IM/2019/038 - Hansini Abeygunawardhana
3. IM/2019/042 - Hafsa Aarifeen
4. IM/2019/054 - Dishan Sandasara
5. IM/2019/113 - Adshayan Rajendran

Contents

Background of the project

Distributed system, a number of computers or servers collaborate through a network to solve a problem or complete a task. These computers are linked by a communication network and are referred to as nodes in distributed systems. A distributed system's nodes may each have their own processing power, memory, and storage, and they may cooperate to accomplish a single objective. Numerous advantages, including enhanced performance, scalability, fault tolerance, and reliability, are promised by distributed systems. Large-scale web services, cloud computing platforms, social networks, financial systems, scientific computers, and a wide variety of other real-world applications all use them. Geographically dispersed and autonomous, distributed systems operate. A food ordering distributed system is a system that enables users to order food from multiple restaurants through a network of interconnected devices. The system is typically composed of multiple components, including user interfaces, databases, and servers that work together to process and fulfill food orders. In a distributed system, these components are spread across multiple locations, allowing for scalability and improved performance. The project typically involves the development of a user-friendly interface for customers to place orders, a database to store customer and restaurant information, a payment gateway for secure transactions, and a server for processing orders and communicating with the different components of the system. The system may also include features such as order tracking, restaurant management, and customer relationship management.

Requirement

The Online Food Order System aims to provide a convenient and efficient way for users to place food orders. The system offers several benefits to users. Firstly, it offers a user-friendly interface for users to browse the menu, select their desired food items, and customize their orders based on preferences or dietary restrictions. This ensures that users can have a personalized dining experience.

Secondly, the system simplifies the order placement process by streamlining the steps involved. Users can easily navigate through the menu, add items to their cart, and proceed to checkout with

a few simple clicks. This saves time and effort for users, making the overall ordering process quick and efficient.

Furthermore, the system ensures the accuracy and reliability of orders. Users can review their order details before finalizing the purchase, reducing the likelihood of errors or misunderstandings. This helps in maintaining customer satisfaction and minimizing order discrepancies.

Additionally, the Online Food Order System facilitates secure online payments, providing users with a seamless and convenient transaction experience. It should support various payment methods, such as credit cards, debit cards, or digital wallets, to cater to the diverse preferences of users.

Moreover, the system should provide real-time updates on the order status, such as confirmation, preparation, and delivery. Users should receive notifications or have access to a tracking feature that allows them to monitor the progress of their order. This enhances transparency and keeps users informed throughout the entire process.

Lastly, the system should prioritize user data privacy and security. It should implement appropriate measures to protect user information, including encryption of sensitive data and adherence to privacy regulations.

In conclusion, the Online Food Order System offers a user-friendly and efficient platform for users to conveniently place food orders. By providing a seamless ordering experience, ensuring order accuracy, and prioritizing user security, the system aims to enhance customer satisfaction and streamline the food ordering process.

Technologies

Azure (Cloud Computing Platform)

Our choice of Azure as the cloud computing platform brings several advantages to our distributed system application. Azure offers a wide range of infrastructure and platform services that enhance scalability, reliability, and security. We leverage Azure's virtual machines and virtual networks to deploy and manage our application infrastructure securely. Platform services like Azure Container Instances (ACI) and Azure Kubernetes Service (AKS) enable us to orchestrate and scale our

containers efficiently. Additionally, Azure's global reach ensures high availability and reliable delivery of our food ordering system to users worldwide.

PHP (Hypertext Preprocessor)

In our distributed system application, PHP serves as the backbone of our server-side logic. With its versatility and extensive library support, PHP enables us to handle user requests, process data, and generate dynamic web content. We leverage PHP's capabilities to implement essential features such as user authentication, menu management, order processing, and integration with external APIs. By harnessing PHP's power, we ensure seamless communication between the frontend and backend, delivering a smooth and interactive user experience.

MySQL (Relational Database Management System)

As a crucial component of our application, MySQL provides a reliable and efficient solution for storing and retrieving data. With our expertise in MySQL, we design optimized database structures, ensuring efficient data storage and retrieval. We utilize its powerful query language to manage menu items, user profiles, order details, and other critical data. By leveraging MySQL's features such as transactions and indexing, we guarantee data integrity and maximize query performance. Our proficiency in MySQL enables us to handle complex data relationships and deliver efficient database operations for our distributed system.

Docker (Containerization Platform)

Docker plays a pivotal role in our distributed system architecture, enabling us to package our application and its dependencies into lightweight and portable containers. With Docker, we achieve consistency across different environments, simplifying deployment and management. By containerizing our application components, including PHP, database, and other services, we ensure seamless integration and portability. Docker facilitates scalability, enabling us to scale our application as demand grows. It also streamlines the deployment process and ensures efficient

resource utilization, enhancing the overall performance and manageability of our distributed system.

Distributed system Architecture

Client-server architecture is a computing model where tasks and responsibilities are divided between a client and a server. The client, typically a user's device or application, initiates requests for services or resources, while the server, a powerful computer or system, processes those requests and provides the requested services. Communication between the client and server occurs over a network using protocols such as HTTP or TCP/IP. The client sends a request specifying the desired service, and the server responds with the requested data or performs the required operation. This architecture allows for scalability, with multiple clients connecting to a single server or a cluster of servers, enabling efficient workload distribution. The centralized nature of client-server architecture also facilitates centralized management and control of resources, data, and security. It is a widely used model in applications and systems such as web and mobile applications, databases, cloud computing, and distributed systems, offering scalability, flexibility, and efficient management of complex system.

Deployment Architecture

Deployment Architecture for Distributed Online Food Ordering Game follows.

Frontend

The front end of our online food ordering distributed system is designed to provide customers with an appealing and intuitive interface, ensuring a seamless and engaging user experience. It incorporates user interfaces, menus, forms, and interactive elements to facilitate easy navigation and interaction. With responsive design, the front end adapts to different devices and screen sizes, providing an optimal viewing experience across desktops, laptops, tablets, and mobile phones. It showcases menus from various restaurants in an organized and visually appealing manner, allowing customers to explore food options, view descriptions and prices, and customize their orders. Additional features like search functionality and user account management further enhance the user experience.

our online food ordering system focuses on delivering a visually appealing, intuitive, and responsive interface. It ensures that customers can effortlessly browse menus, customize orders, and navigate through the system with ease. With features like search and account management, the front end enhances convenience and engagement, making the overall ordering process seamless and enjoyable for customers.

Backend

The back end of our online food ordering distributed system, built with PHP and MySQL, handles the server-side operations and data management. PHP is used for implementing the back-end logic, while MySQL is used as the database for storing and managing application data. The back-end processes user requests, performs computations, and interacts with the database to retrieve or modify data. It handles functionalities such as order processing, user authentication, and data storage. By using PHP and MySQL, you can create a scalable and efficient back-end system for your online food ordering application.

Database

As the developer of our distributed food ordering system, MySQL plays a vital role as our chosen database management system. With its strong reputation for performance and scalability, MySQL allows us to efficiently store and manage the application's data.To leverage the power of MySQL, I have carefully designed and implemented a structured database schema that reflects the entities and relationships within our food ordering system. This includes tables for customers, restaurants, menus, orders, and more, ensuring that data is organized and easily accessible.In our PHP code, I have integrated MySQL functionality using appropriate functions and libraries. This enables seamless communication between the back end and the MySQL database. By establishing a connection to the MySQL server, we can execute SQL queries to retrieve, modify, or store data as needed. Throughout the system, MySQL facilitates critical operations. For example, when a customer places an order, our PHP code interacts with the MySQL database to store order details, update inventory levels, and manage order statuses. MySQL also supports other important functionalities, such as retrieving customer information, generating reports, and performing data

analysis. As the developer, I have optimized MySQL performance by applying indexing techniques, optimizing query execution, and implementing caching mechanisms. These measures ensure that our database can handle a growing volume of data and deliver efficient responses. Overall, MySQL serves as the backbone of our distributed food ordering system, providing robust data storage, retrieval, and management capabilities. Its scalability, performance, and integration with PHP empower us to build a reliable and efficient system that meets the needs of our users and supports seamless food ordering experiences.

Communication Protocol

In our distributed food ordering system, we have implemented a RESTful API communication protocol to facilitate seamless communication between the front-end, back-end, replicated servers, and database components. RESTful APIs provide a standardized and scalable approach, allowing clients to send HTTP requests to designated endpoints for data retrieval and actions. The back-end server processes these requests, interacts with replicated servers and the database, and sends back appropriate HTTP responses. This communication protocol ensures efficient and reliable data exchange, enabling smooth order placement functionality within our system.

Networking Infrastructure

In our food ordering project, we have established a networking infrastructure that encompasses key components for reliable communication and scalability. This includes a Virtual Private Cloud (VPC) that provides a secure and isolated network environment. Within the VPC, we have implemented a load balancer to distribute incoming traffic among multiple web/application servers deployed in different availability zones. This ensures efficient resource utilization and high availability. Our MySQL database server is set up within the VPC, offering secure storage for application data and handling read and write operations effectively. Private subnets and security groups are utilized to isolate internal components and control access to specific resources. Secure remote access to the network infrastructure is enabled through a VPN or SSH mechanism. We have integrated monitoring and logging tools to track the health, performance, and security of the network infrastructure, facilitating proactive issue detection and resolution. This networking

infrastructure provides a strong foundation for secure communication, scalability, and efficient resource management, ensuring the reliable and seamless operation of the food ordering system.

Deployment

To deploy the food ordering distributed system, we followed a systematic process to ensure smooth deployment and configuration.

First, we provisioned the necessary infrastructure in a cloud environment, such as Microsoft Azure. This involved creating virtual machines (VMs) for the master and worker nodes, configuring networking settings, and allocating appropriate resources to each VM.

Next, we installed Docker on each VM to enable containerization. Docker provides a platform to package the application and its dependencies into containers, ensuring consistent deployment across different environments.

We then containerized each component of the food ordering system by creating Docker images. These images contained the necessary configurations and dependencies to run the application, including the PHP application, MySQL database, and any other required services.

To manage the distributed deployment, we set up a Docker Swarm cluster. The master VMs were designated as swarm managers, while the worker VMs joined the swarm as worker nodes. This allowed us to orchestrate and scale the application across multiple nodes.

Using a Docker Compose file, we defined the services, networks, and volumes required by the application. The Compose file specified the desired configuration, such as the PHP image, exposed ports, environment variables, and volume mappings.

With the Docker Swarm cluster and Docker Compose file in place, we deployed the food ordering system by running the Docker Compose command on the swarm manager. This initiated the creation of service replicas across the worker nodes, ensuring redundancy and scalability.

To distribute incoming traffic among the replicated services, we utilized a load balancer. This load balancer acted as a central entry point, forwarding client requests to the appropriate containers, providing high availability and optimized resource utilization.

We established a CI/CD pipeline to automate the deployment process. This involved using tools like Git for version control, Jenkins for continuous integration, and Docker Registry or Docker Hub for image storage. Whenever changes were pushed to the repository, the CI/CD pipeline triggered automated testing, building new Docker images, and deploying updates to the swarm.

By following this deployment approach, we ensured that the food ordering distributed system was effectively containerized, orchestrated, and scaled across multiple nodes. This allowed for efficient resource utilization, improved availability, and streamlined deployment and management processes.

Key challenges

Transparency

Transparency was a fundamental principle we embraced throughout the development of our distributed food ordering system. To achieve this, we implemented various strategies to ensure clear communication and documentation. First and foremost, we fostered open and transparent communication among team members. Regular meetings, stand-ups, and brainstorming sessions were held to discuss progress, challenges, and decisions made during the development process. This encouraged collaboration and ensured that everyone was informed about the project's status. In addition to communication, we placed a strong emphasis on comprehensive documentation. We documented the system's design, architecture, and components in detail. This included clear explanations of the system's functionalities, interfaces, and data flow. By documenting our decisions, we ensured that all team members and stakeholders had access to the same information, promoting transparency and shared understanding. Moreover, we made our development process transparent by leveraging version control systems, such as Git. This allowed us to track changes, review code, and maintain a detailed history of modifications made to the system. Through the use of branches, pull requests, and code reviews, we ensured that all code changes were visible, providing transparency and enabling collaboration among the development team. Overall, by prioritizing clear communication, comprehensive documentation, and transparent development practices, we successfully achieved transparency in the development of our distributed food ordering system. This transparency not only fostered a shared understanding among team members

but also allowed stakeholders to have a clear view of the system's progress and decision-making process.

Availability

Achieving high availability was a top priority during the development of our distributed food ordering system. To ensure continuous access to our application, we implemented several key strategies. Firstly, we focused on redundancy by deploying multiple instances of critical components such as web servers, application servers, and databases. This redundant setup allowed us to distribute the workload across multiple instances, so if one instance failed, the system seamlessly shifted the traffic to other healthy instances, minimizing downtime and maintaining availability. Load balancing played a crucial role in optimizing resource utilization and improving system availability. We employed load balancers to evenly distribute incoming traffic across the available instances of our application. This approach prevented any single instance from becoming overwhelmed and ensured that user requests were efficiently processed, contributing to an enhanced user experience and continuous availability. Additionally, we prioritized fault tolerance in our system design. By leveraging container orchestration platforms like Docker Swarm or Kubernetes, we could monitor the health of containers and automatically restart or replace any failed instances. This proactive approach to handling failures minimized disruptions and improved overall availability by promptly recovering from failures. To monitor the health and performance of our system, we implemented robust monitoring and alerting mechanisms. These tools continuously monitored various metrics such as resource usage, response times, and error rates. In case of any anomalies or issues, real-time alerts were triggered, allowing us to promptly address the problem and ensure high availability. Scalability was also a key consideration. By designing our system to be horizontally scalable, we could add more resources, such as virtual machines or containers, to accommodate increased demand and user traffic. This elastic scalability allowed us to handle peak loads and maintain availability even during periods of high demand. Lastly, we implemented a comprehensive disaster recovery strategy. Regular backups of our databases and critical data were performed, and we had well-defined plans for restoring the system in case of catastrophic failures. This approach ensured that data integrity was preserved, and the system could quickly recover from any unforeseen events, minimizing downtime and maintaining availability.

By combining redundancy, load balancing, fault tolerance, monitoring, scalability, and disaster recovery measures, we successfully achieved a high level of availability in our distributed food ordering system. This allowed us to provide uninterrupted access to our application, ensuring a reliable and seamless experience for our users.

Performance

Performance was a key focus area during the development of our distributed food ordering system. We implemented various strategies to optimize system performance and ensure a fast and responsive user experience. Firstly, we focused on efficient resource utilization. By carefully analyzing the system's requirements, we optimized the allocation of computing resources such as CPU, memory, and storage. This allowed us to ensure that the system had sufficient resources to handle user requests and process data without unnecessary bottlenecks or performance degradation. In addition, we employed caching mechanisms to improve response times and reduce database load. By caching frequently accessed data or query results, we minimized the need for repetitive database queries, resulting in faster response times and improved overall system performance. Furthermore, we optimized database operations by designing efficient database schemas, indexing relevant fields, and optimizing query execution plans. These optimizations reduced the time required for data retrieval and manipulation, resulting in faster database performance and improved application responsiveness. Parallel processing and asynchronous operations were also utilized to enhance performance. By breaking down complex tasks into smaller subtasks and processing them concurrently, we were able to leverage the available computing resources efficiently and reduce the overall processing time. Asynchronous operations allowed us to handle multiple requests simultaneously, ensuring a more responsive user experience. We also implemented performance monitoring and profiling tools to identify performance bottlenecks and optimize system components. By analyzing system metrics, we could pinpoint areas that required improvement and apply optimizations to enhance overall performance. Furthermore, load testing and performance testing were conducted to simulate real-world scenarios and measure system performance under various load conditions. These tests allowed us to identify any performance issues, fine-tune system configurations, and ensure that the system could handle expected traffic without compromising performance. Overall, through efficient resource

utilization, caching mechanisms, database optimizations, parallel processing, performance monitoring, and load testing, we successfully achieved high performance in our distributed food ordering system. These strategies ensured that the system could handle a large volume of requests, provide fast response times, and deliver a seamless user experience.

Reliability

During the development of our distributed food ordering system, we focused on achieving reliability by implementing various strategies. We ensured fault tolerance through redundancy and failover mechanisms, enabling the system to continue functioning even in the event of failures. Robust data backup and recovery mechanisms were in place to maintain data integrity and availability. Real-time monitoring, proactive maintenance, and automated alerts helped us detect and address issues promptly. Rolling deployments and version control ensured seamless updates without disrupting system operation. Rigorous testing at different stages validated the reliability and stability of the system. Overall, these measures ensured a dependable and consistent experience for our users.

Concurrency and Consistency

Achieving concurrency and consistency were critical objectives during the development of our distributed food ordering system. We implemented specific approaches to ensure that multiple users could interact with the system simultaneously while maintaining data consistency across all components. To achieve concurrency, we employed techniques such as multi-threading and parallel processing. By utilizing multiple threads, we were able to handle concurrent user requests and execute tasks in parallel. This allowed the system to efficiently serve a large number of users without compromising performance or responsiveness. In terms of consistency, we relied on distributed database management systems and synchronization mechanisms. We designed our data models and schemas to support ACID (Atomicity, Consistency, Isolation, Durability) properties, which ensured that transactions were executed in a consistent manner. By using distributed databases, data was replicated across multiple nodes, enabling data consistency and minimizing the risk of data discrepancies. We also implemented distributed locks and distributed coordination

protocols to manage concurrent access to shared resources and ensure consistency. These mechanisms helped prevent conflicts and maintained data integrity during concurrent operations. Furthermore, we employed messaging queues and event-driven architectures to facilitate asynchronous communication and maintain consistency across different components of the system. By decoupling components and enabling asynchronous processing, we ensured that actions and updates performed in one part of the system were propagated consistently to other relevant components. In summary, through the use of multi-threading, parallel processing, distributed databases, synchronization mechanisms, distributed locks, coordination protocols, messaging queues, and event-driven architectures, we achieved both concurrency and consistency in our distributed food ordering system. These approaches allowed multiple users to interact with the system concurrently while maintaining data integrity and consistency across all components.

Scalability

Scalability is a crucial aspect of our distributed food ordering system built with PHP, focusing on the placement of orders. It involves designing the system to efficiently handle a growing number of users, orders, and resources. By implementing strategies such as load balancing and leveraging technologies like Azure, we ensure even distribution of workload and optimal performance, even during high traffic periods. Overcoming challenges related to increased load and evolving requirements, our aim is to create a scalable system that accommodates multiple players, enhances concurrency, maintains consistency, and ensures fault tolerance for a seamless user experience.

Security

Security is a paramount concern in our distributed food ordering system developed using PHP, particularly in the context of placing order functionality. We prioritize implementing robust security measures to safeguard user data, prevent unauthorized access, and ensure the integrity of transactions. To ensure data security, we employ various techniques such as encryption and secure data transmission protocols (such as HTTPS) to protect sensitive information during communication between the client and server. User authentication mechanisms, such as username/password or token-based authentication, are implemented to verify the identity of users

and prevent unauthorized access to their accounts. Protection against common security threats, including cross-site scripting (XSS) and SQL injection attacks, is a priority. We employ input validation and sanitization techniques to mitigate these risks and ensure that user-supplied data is properly handled and does not compromise the system's security. Additionally, we implement secure coding practices and follow industry standards to minimize vulnerabilities in the codebase. Regular security audits and penetration testing are conducted to identify and address potential weaknesses or vulnerabilities in the system. Proper access controls are enforced to ensure that users have appropriate permissions based on their roles and privileges. This prevents unauthorized users from accessing sensitive functionalities or performing actions beyond their authorized scope. Overall, security is a top priority in our distributed food ordering system. By implementing strong encryption, robust authentication mechanisms, secure coding practices, and regular security audits, we aim to provide a secure environment for our users, safeguard their data, and ensure a trustworthy ordering experience

Limitations

While developing our distributed food ordering system with PHP and placing order functionality, we must consider a few limitations. Firstly, effectively handling increased load and meeting growing needs can pose challenges, requiring careful planning and implementation. Adapting to changing workload patterns and requirements demands continuous evaluation and scalability strategies.

Another limitation is the absence of multi-player functionality in our current system. Expanding it to support multiple players would improve concurrency and consistency, but it also introduces complexities in managing concurrent updates and ensuring data consistency across shared game states.

Additionally, we should be mindful of potential issues related to middleware dependency. If the middleware component crashes, it can significantly impact the system's availability, reliability, and performance. Implementing proper fault tolerance mechanisms, such as redundancy, replication, fault handling, and constant monitoring, is crucial to mitigate the risk of middleware failures and ensure system stability.

Understanding and addressing these limitations is essential to enhance the scalability, performance, and reliability of our distributed food ordering system. By identifying and overcoming these challenges, we can build a robust and adaptable system that caters to the growing needs of users and provides an exceptional ordering experience.

****