NEU Study- Meeting Room Reservation System
Group2 Executive Report

Ying Tuo, Xi Shen, Yuqin Luo, Huixin Huang

Khoury College of Computer Sciences
Northeastern University, San Jose, CA 95138

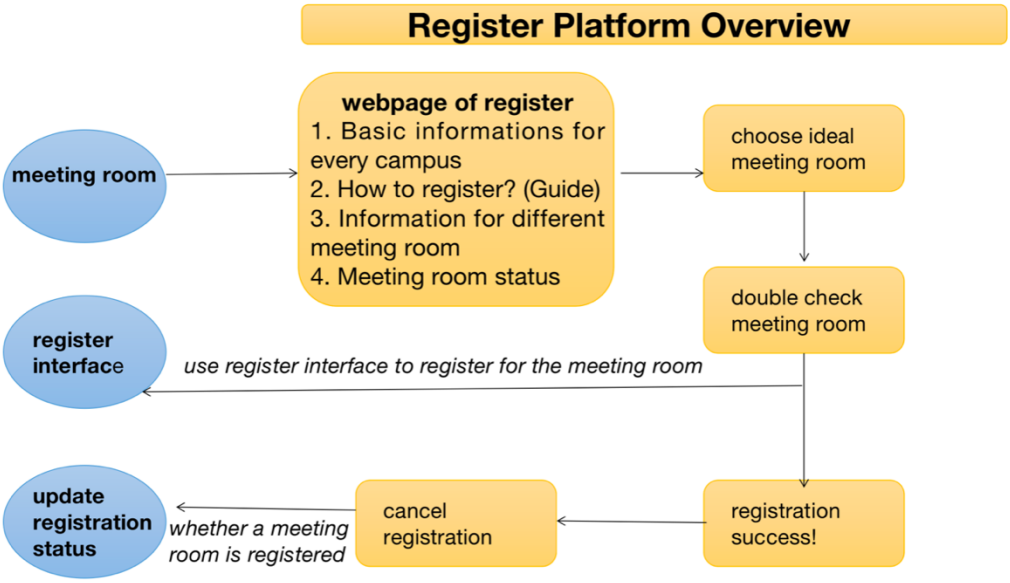
ABSTRACT

NEU Study - Meeting Room Reservation System is a system designed for Northeastern University students to reserve meeting rooms. When students want to go to the library to study or discuss with their classmates in a meeting room, students often need to go to several rooms to find an available one. This system provides a meeting room reservation function, which allows students to easily find available rooms and make reservations so that they do not need to waste their valuable time finding rooms. We used microservice structure, consensus algorithm (Raft), message queue, digital signatures (MD5) and other techniques to develop it as a distributed management system. This reservation system has great performance in highly available, reliable, resilient, easy to use, and efficient.

Keywords: Distributed System, Microservice, Raft, Message Queue, MD5

1 INTRODUCTION

NEU Study - Meeting Room Reservation System is a comprehensive distributed management system of the administrator platform and user platform.
In the administrator platform, the staff of Northeastern University can check the status of every meeting room, for example, whether a meeting room is reserved or available for a special event. The administrator can also add, check, update or delete (CRUD) the information of meeting rooms, including the campus it belongs to and the meeting room size.
The staff of Northeastern University can check three modules of this administrator system, including Data Administration Module, Meeting Room Administration Module, and Registration Administration Module. For the Data Administration Module (*P1. architecture overview diagram*), it maintains some constants, like campus data, the style of meeting rooms, and the identity of the person who registered for the meeting room. For the Meeting Room Administration Module, the administrator can check the status, and whether it is reserved. For the Registration Administration Module, the administrator can check the current overall information, which meeting room is booked, and the information about the meeting room.

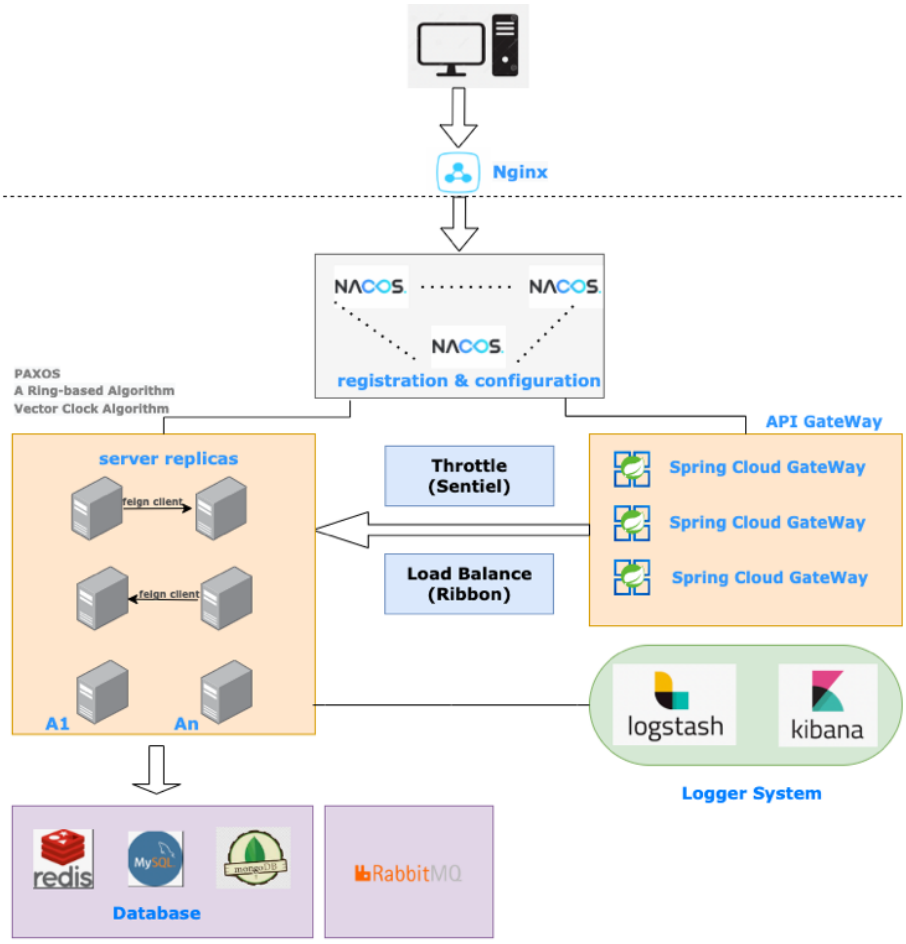


P1. architecture overview diagram

In the User Platform, users(students/teachers) can login to their accounts and click on the icons to check campus and meeting room information and register for meeting room. If they have other plans after the registration, they can also login and cancel the reservation.
We used microservice structure, consensus algorithm (Raft), message queue, digital signatures (MD5) and other techniques to develop it as a distributed management system.

2 DESIGNS

As for the design of the project, on the client-side, we first need to register for the service. The register center is composed of Nacos (based on Raft algorithm), which is a dynamic naming and configuration service. We can get a list of services by Spring Cloud Gateway. Spring Cloud Gateway is mainly responsible for routing to solve the CROSS-ORIGIN issues. Based on Nacos, we can also get a distribution of load balance cluster. We use MySQL and MongoDB as microservice databases (repositories). During the development, we used Github for managing code versions and created a distributed management system.
We designed five microservices for this system. Manage Service (port: 9998) is developed for the campus admin. Campus Service (port: 8201) is a central campus information management service. CMN Service (port: 8202) is like a dictionary service that maintains the codes and their corresponding information. User Service (port: 8203) is responsible for managing the users'(student) information. Order Service (port: 8206) is responsible for maintaining the schedule-order records. Details about these services is described in Section 3.1.



P2. Architecture Design

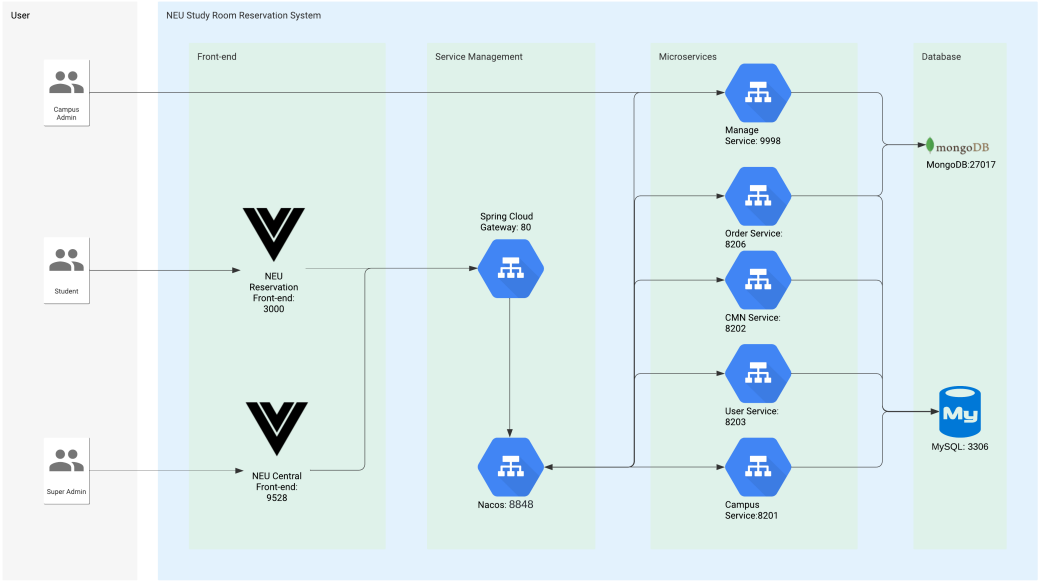
3 METHODOLOGIES

3.1 Microservice

Microservices are an architectural and organizational approach in the software development and distributed system field. In the microservice architecture, the software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams. Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

We designed five microservices for this system. Manage Service (port: 9998) is developed for the campus admin (NEU Silicon Valley Campus is our dummy data in this project). The NEU SV admin can add, check, update, and delete (CRUD) the campus information, room information, and booking information in this microservice. Campus Service (port: 8201) is a central campus information management service where the super admins have access to all the campus information and is responsible for verifying the identification of campus admins who would like to access its corresponding campus database. CMN Service (port: 8202) is like a dictionary service that maintains the codes and their corresponding information, for example, 10001

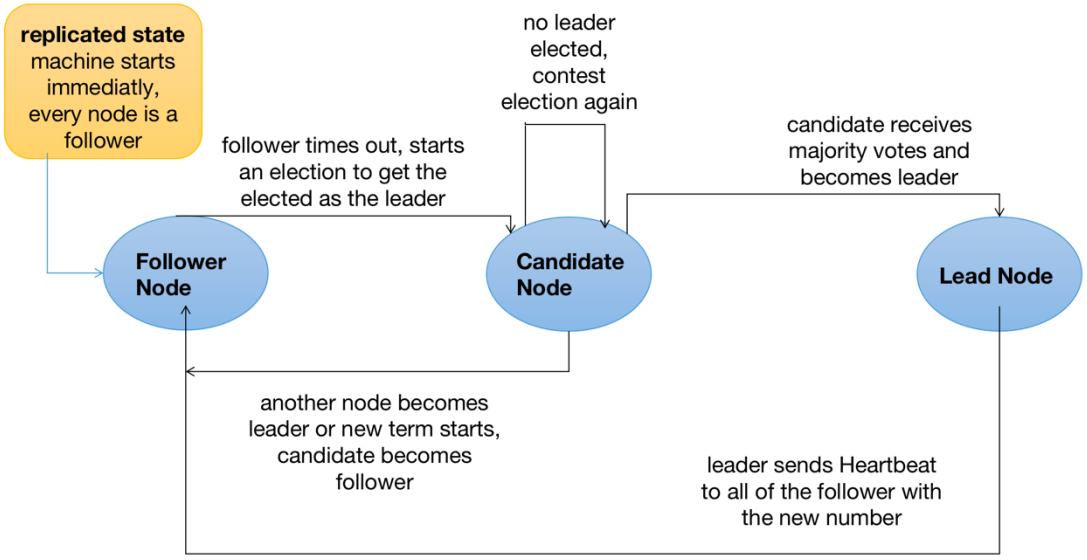
stands for the West Coast, US, Area, where there are Silicon Valley campus, San Francisco campus, and Seattle Campus. User Service (port: 8203) is responsible for managing the users'(student) information, like the students' ID, student password, etc. Order Service (port: 8206) is responsible for maintaining the schedule-order records. When a student submits a scheduling order to our system, this student will successfully reserve the room if the order service can process this scheduling order.



P3. NEU-Study Microservice Architecture Design

3.2 Nacos and Raft and Comparison with Paxos

We learned about Paxos this semester in CS6650. During our research on other similar consensus algorithms, we found Raft is also widely used in the industry.
The Raft algorithm divides the Server into 3 states, or roles. **Leader**: Responsible for Client interaction and log replication, there is at most one in the system at the same time. **Follower**: passively responds to request RPC, never actively initiates request RPC. **Candidate**: a temporary role that only exists in the leader election phase. If a node wants to become a leader, it will initiate a voting request and become a candidate at the same time. If the election is successful, it becomes a leader, otherwise, it returns to followers.



P4. Raft Algorithm

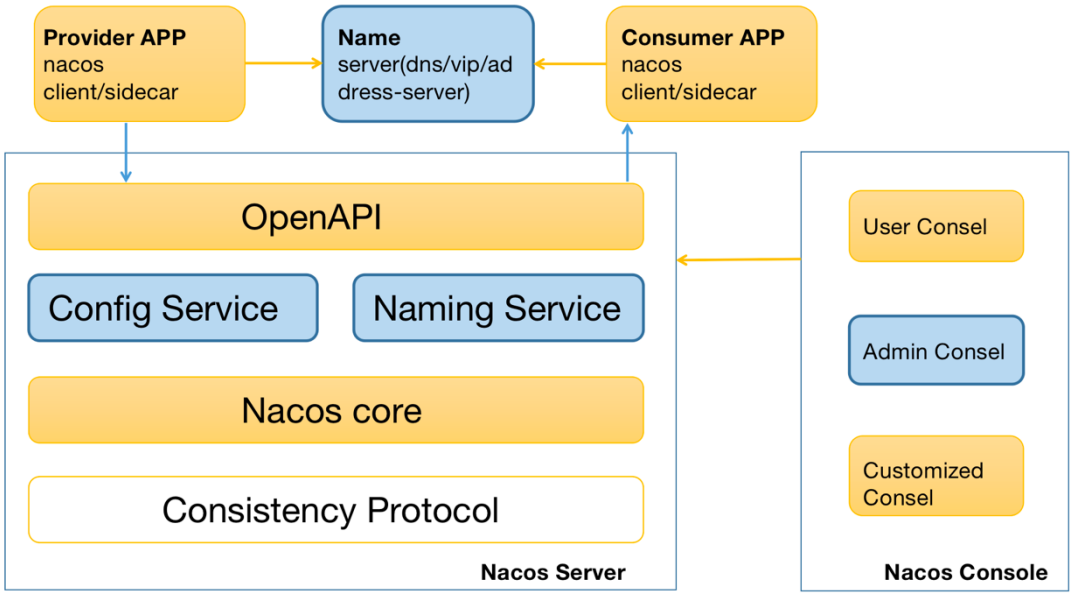
In Raft, the problem is broken down into: leader election, log replication, security, and membership change. The replicated state machine is implemented by replicating the log: Log: Each machine saves a log, which comes from the client's request and contains a series of commands. State Machine: A state machine executes these commands in sequence. Consistency model: In a distributed environment, the logs of multiple machines are guaranteed to be consistent, so that the state played back to the state machine is consistent. We compare the Raft and Paxos in Table 1, which implies the reason why we use Raft.

Table 1 Comparison with Raft and Paxos

	Raft	**Paxos**
Leader	Strong leader	Weak leader
Voting rights for the leader	Have a replica of the latest committed log	Arbitrary replica
Log replication	Guaranteed continuity	Allow voids
Log submission	Push forward the commit index	Asynchronous commit messages

After we decide to use Raft algorithm, we researched and found Nacos is a good framework to help us implement the functionalities of Raft. Nacos is an open-source project launched by Alibaba, which is a dynamic service discovery, configuration management, and service management platform that makes it easier to build cloud-native applications. The CP consistency protocol implementation of Nacos is based on the simplified Raft CP consistency. When the Nacos server starts, it will call the RaftCore.init() method through the RunningConfig.onApplicationEvent() method. Nacos officially supports AP and CP consistency protocols in 1.0.0. The CP consistency protocol implementation is based on the simplified Raft CP consistency

(Distributed CAP theorem: C data consistency, A service availability, P partition fault tolerance).



P5. Nacos Basic Architecture

In this project, we choose the Nacos as the registration center because it not only provides the functions of the registration center, but also the function of the configuration management center, and its products are verified in the production environment. Nacos provides the following four functions: Service discovery and service health monitoring. Dynamic configuration service. Dynamic DNS Service. Services and their metadata management. The function of its management interface is also very good, and it also has its own rights management function. For the microservices in our project, they are registered in Nacos, and also can call other microservices via Nacos.

3.3 Message Queue and RabbitMQ

Indirect Communication plays a role of great importance in a distributed system. Message queues (distributed message queues) are a further important category of indirect communication systems. Message queues provide a point-to-point service using the concept of a message queue as an indirection, achieving the desired properties of space and time uncoupling. They are point-to-point in that the sender places the message into a queue, and it is then removed by a single process. A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures. Messages are stored in the queue until they are processed and deleted. Each message is processed only once, by a single consumer.

In our project, we applied message queues with RabbitMQ for the communication of each microservices. For example, one of our microservices, *service_order* sends

messages to the database by RabbitMQ. Queues in RabbitMQ are ordered collections of messages. Messages are enqueued and dequeued (delivered to consumers) in the FIFO manner (FIFO ordering is not guaranteed for priority and shared queues).

3.4 Security, Digital Signature and MD5

Strong digital signatures are an essential requirement for secure systems. We use digital signatures in order to make the information authentic, unforgettable, and non-repudiable. Digital signatures depend upon the binding of a unique and secret attribute of the signer to a document.

Digital signing is an electronic document or message M that can be signed by a principal A through encrypting a copy of the message with a key Ka and attaching it to a plain text copy of M and A's identifier. The signed documents consist of: M, A, M[Ka]. The signature can be verified by a principle that subsequently receives the document to check that it was originated by A and that its contents, M, have not subsequently been altered.

In our project, we used the secure digest functions MD5. The MD5 algorithm uses four rounds, each applying one of four nonlinear functions to each of 16 32-bit segments of a 512-bit block of source text. The result is a 128-bit digest. MD5 is one of the most efficient algorithms currently available. We just build a basic MD5 class to implement this encoding process.

In the microservices of our project, we implement the MD5 for the *campus_management servic*e (campus admin). In the super admin's database, each campus has its own *sign_key*. If the campus admins would like to add, check, update or even delete (CRUD) the corresponding campus information, they need to encrypt its sign_key (private key) and then send it to the super admin (central campus information management system) for verification. If the encrypted sign_key is correct, the campus admin will get access to its corresponding campus information and the request will be processed.

4 CONCLUSIONS

During developing this distributed system, NEU Study－Meeting Room Reservation System, our group get deep understanding of the distributed system and learn about how to implement the techniques and algorithm into an industrial project, like the microservice, message queue, consensus algorithm (Paxos and Raft), digital signature (MD5), etc.

The NEU Study－Meeting Room Reservation System finally has great performance in the following aspects. Highly Available, we build six microservice for this reservation system and plan to deploy them in the cloud if we have more time, which can improve the system's bandwidth. Reliable, we considered diverse corner cases with handling different error expectation and the system have minimum downtime. Resilient, our system is resilient to error. Easy to use, we bridge the frontend and backend using RESTful APIs, and we designed and developed a friendly UI for users

(students) to book their meeting room. Efficient, our system has low latency and process time since we use the microservices structure.
As for the future research field, firstly, we can deploy these microservices to Kubernetes (or other cloud services like AWS), so that we can have more replicas to support the operation of our system. Besides, we just implemented a simple user login functionality, we can develop email registration and Google login functionalities in the future.

REFERENCE

[1] Coulouris, George, Jean Dollimore, and Tim Kindberg. "Distributed Systems: Concepts and Design Edition 5." System 2.11 (2012): 15.

[2] Lamport, Leslie, “Paxos made simple.” ACM SIGACT News {Distributed Computing Column} 32, 4 (Whole Number 121, December 2001) (2001): 51 - 58

[3] Lamport, Leslie, Robert Shostak, and Marshall Pease. “The Byzantine general problem.” Concurrency: the Works of Leslie Lamport,. 2019.203-226

[4]https://www.alibabacloud.com/blog/paxos-raft-epaxos-how-has-distributed-consensus-technology-evolved_597127

[5] Bakshi, Kapil. "Microservices-based software architecture and approaches." 2017 IEEE aerospace conference. IEEE, 2017

[6] Ongaro, Diego, and John Ousterhout. "The raft consensus algorithm." (2015): 54.

[7] Hong, Xian Jun, Hyun Sik Yang, and Young Han Kim. "Performance analysis of RESTful API and RabbitMQ for microservice web application." 2018 International Conference on Information and Communication Technology Convergence (ICTC). IEEE, 2018.

[8] Rivest, Ronald. The MD5 message-digest algorithm. No. rfc1321. 1992.