Software Practice of Orderly Charging Information Communication System

Weiguo Wang, Tao Li and Yu Pan
Beijing Smart-chip Microelectronics Technology Ltd., Changping District, Beijing 102200, China

Abstract. At present, SGCC actively implements the national New Energy Strategic Policy, vigorously promotes the construction of electric vehicle charging piles, and actively guides electric vehicles to participate in orderly charging through multiple ways. Combined with the construction objectives and requirements of SGCC for orderly charging business, based on the functions of the original charging pile operation management system, our company (Beijing Smart-chip Microelectronics Technology Ltd.) has realized the orderly charging system of charging pile through structural transformation and function upgrading. The system is realized by distributed and microservice architecture mode, which can support large-scale charging piles fast access. This paper describes the orderly charging system of charging pile and the practice of distributed and microservice architecture in the system.

Keywords: Microservice; Spring cloud; Orderly charging.

1. Introduction
At present, the charge pile operation management system[1] developed by our company can realize many functions, such as reservation charging, personnel vehicle navigation, charging information remote viewing and intelligent control, charging bill mobile payment, etc., which greatly improves the user convenience. However, with the development of the system platform, there are more and more kinds and quantities of terminal equipment such as electric vehicle charging pile, and the platform is developed based on the traditional vertical application architecture mode. Therefore, the response speed of the system becomes more and more slow, and it is very difficult to upgrade the software function.

Today the widespread practice and successful experience of distributed and microservice architecture at home and abroad bring hope to solve the above problems. Distributed and microservice architecture has the advantages of flexible selection of development technology stack, independent expansion and on-demand scaling of various services, loose coupling between services and good isolation in case of failure, and high utilization rate of system hardware resources.

On the basis of the main functions of the operation and management system of charging piles, the system can realize the synchronous intelligent coordination between the real-time load of the power grid in the Zone Area and the output power of the orderly charging piles in the Zone Area by upgrading the software and hardware of the existing charging piles and implementing the distributed and micro service software architecture. The system can make full use of the idle resources of the power grid and keep the stable operation of the power grid in the Zone Area to the maximum extent. This paper will introduce the whole structure and basic work flow of the orderly charging system, and elaborate the core part of the system - the front host software architecture design and specific implementation of the data communication function.
2. Introduction to the Overall Structure of Orderly Charging

The comparison of the overall hierarchical structure between the charging pile operation management system and the orderly charging system is shown in the following figure:

![Figure 1. Structure comparison chart.](image)

a) the charging pile operation management system; b) the orderly charging system

On the left side of the figure above is the charging pile operation management system, and on the right side of the figure above is the orderly charging system.

In order to achieve orderly charging, a charging control module must be installed inside each charging pile. The charging control module has the functions of controlling charge pile in the down direction and communicating with edge aggregation controller in the up direction. After the new charging pile is enabled, it will periodically and actively report its heartbeat message and working condition data to the edge aggregation controller through the control module, and then the edge aggregation controller will communicate with the front host. A single edge aggregation controller can aggregate the uplink and downlink data of several control modules. The communication process among charging pile, charging control module and edge aggregation controller is simple, and there is no risk of performance pressure. Mobile app is the only entrance for users to participate in orderly charging activities and interact with charging piles. The data interaction and command control between mobile app and charging pile can only be realized by front host program via edge aggregation controller and charging control module.

The front host program is the data communication centre of the whole software system, which is used to coordinate the data interaction among its internal services, mobile app and edge aggregation controller. The front host program uses the heartbeat mechanism to maintain a long TCP connection with the edge aggregation controller to be accessed, and processes and responds to the up and down instructions and data from the mobile app and from the edge aggregation controller.

The front host program mainly includes two parts: distributed communication link connection management and related programs based on microservice architecture.

3. Distributed Communication Link Connection Management

In order to ensure the communication efficiency, the long link communication mode based on TCP is adopted between the edge aggregation controller and the front host program in the orderly charging system. The communication link between the edge aggregation controller and the front host program is managed in a distributed cluster mode, as shown in the following figure:

![Figure 2. Schematic diagram of link connection management.](image)
Each netty node server will go to zookeeper to register a temporary node after startup. In case of node failure or node logoff, the temporary node will be automatically deleted by zookeeper. Zookeeper is used for netty cluster management [2].

After the edge aggregation controller is started, it will register with zuul gateway. Zuul gateway queries the list information of available netty servers in zookeeper, and returns the information of one of the netty servers to the edge aggregation controller through a certain load balancing algorithm [3]. Next, the edge aggregation controller will establish a long TCP link with the netty server, and cache it in the redis cluster after the connection is established. The heartbeat, data and control messages between the subsequent edge aggregation controller and the netty server are all communicated through this link [4].

4. Microservice Infrastructure and Deployment Encapsulation Tool

The mobile app, the front host program and the services inside the front host adopt the RESTful HTTP access mode. The front host program mainly uses Spring Cloud technology stack[5] and multiple excellent open source components[6] to realize microservices. Software developers can focus on specific businesses without paying too much attention to the overall architecture of microservices, so it can minimize the difficulty of application development and shorten the development cycle. At the same time, we also use Gitlab, Jenkins and Docker container technology to realize distributed code management, code continuous integration, service encapsulation and rapid deployment. As shown in the figure below:

![Figure 3. Schematic diagram of Microservice infrastructure and deployment encapsulation tools.](image)

The dotted line in the figure above is the schematic diagram of the front host program service composition.

The microservice infrastructure and deployment encapsulation tools used in this paper mainly include the following components:

4.1. Eureka

Eureka is the service registration and discovery component in the spring cloud microservice overall solution, which is responsible for the registration, discovery and state synchronization of services, and is the information hub for successfully completing the call between services in the microservice framework.

4.2. Zuul

Zuul is the API gateway and API service management component in the overall spring cloud microservice solution. As the only access to system services, this component shields all the internal complexities, and simplifies the communication between client implementation and microservice applications. This component supports cluster deployment, so it can easily handle ultra-high concurrency, support horizontal scaling, and provide external authentication, security, traffic control, logging, protocol conversion, access routing, load balancing and other functions [7].

4.3. Netflix hystrix

Netflix hystrix is a service governance and monitoring component. Hystrix is a tool class library that implements timeout mechanism and breaker mode. The service circuit breaker can quickly handle the fault in the process of service call by configuring the rules of fusing and degradation, so as to ensure the
service call can respond quickly and avoid the avalanche effect. At the same time, Dashboard and Turbine are used to display the real-time monitoring data and results of multiple services in the micro service cluster.

4.4. Apollo
Apollo is an open-source configuration management centre component released by Ctrip. Com, which can centrally manage the configuration of different environments and clusters of applications. After the configuration is modified, it can be pushed to the application end in real time, and has the characteristics of standard permissions, process management, etc. This component can realize public information configuration through Gitlab, SVN and local file storage. Its function and performance are obviously better than that of Spring Cloud Config, and the component has been strictly verified by the production environment.

4.5. Tools
In the development stage, developers submit business code to Gitlab remote code repository from time to time. Jenkins, a continuous integration tool, will periodically pull the specified project code from Gitlab repository according to the preset and automatically complete the construction to generate a test environment based on Docker container. The testers work in the test environment. After repeated iterations, the stable project source code is obtained after the test. Finally, the docker image of the project is obtained through Jenkins and Docker again. All microservices are deployed and run online in Docker container mode. Using Docker container technology, we can encapsulate the services implemented by different programming languages and their special running environment, and take the encapsulated image as the delivery. The implementers don't need to spend much energy and time to deploy the software running environment to solve all kinds of strange problems caused by different environments.

5. Specific Application of Microservice Architecture in Front Host Program
The distributed and microservice architecture is used to realize the front host program, aiming to optimize the overall architecture of the system to achieve high performance, high availability and elastic scaling. By dividing the single application into multiple microservice modules that can run independently, the coupling degree between system modules can be minimized[8], the rapid code development and service node deployment can be realized, the time cost of operation and maintenance can be greatly reduced, and the overall expansion ability of the system can be significantly improved.

The front host program can be divided into six microservices according to business functions, as shown in the figure:

**Figure 4.** Internal microservice architecture of the front host program.

These six microservices include two interface services and four internal services. Interface services include interface service for app, communication service for edge aggregation controller, and internal
services include orderly charging strategy service, protocol message parsing and encapsulation services, data storage service, and exception handling service.

The edge aggregation controller communication service manages TCP long link communication with the edge aggregation controller. The uplink data received by the edge aggregation controller communication service will first be put into the RabbitMQ message queue[9], and the protocol message parsing and encapsulation service will always listen to the message queue. After listening to a new message arrival, the message will be parsed. After the parsing is completed, the data storage service will write or update the parsed data to the Redis cache. The data to be warehoused is stored in the MySQL database. If the app interface service needs some data, just go to Redis to get it.

The user initiates an orderly charging request through the app interface service. The app interface service will call the protocol message parsing and encapsulation service, which will then send a request message to the orderly charging strategy service. According to the pre-set policy generation rules, the corresponding orderly charging plan is generated according to the content of the request message and returned to the protocol message parsing and encapsulation service. It is then placed in the RabbitMQ message queue, and when the edge aggregation controller communication service listens to new message arrival, it will send them to the corresponding edge aggregation controller.

In the process of orderly charging, if the protocol message parsing and encapsulation service finds any data exception after parsing the uplink data, it can call the orderly charging strategy service again to generate a new charging plan dynamically and send it.

If an exception occurs during service execution and invocation, it is handled by the exception handling service and recorded in the log.

It should be pointed out that some load balance strategies are used in the calls between services, which can ensure the smooth access of a large number of charging piles to the system and the normal operation and high availability of real-time services in the system in the future.

6. Conclusion
With the continuous improvement of the system architecture and functions, the access of a large number of charging piles following the general protocol can be easily handled only through the expansion service node in the future. Therefore, the practical implementation of the distributed and microservice architecture in the orderly charging system is of great positive significance for improving the stability and reliability of the system, subsequent system maintenance, and stable upgrading and expansion.

This paper discusses the practical application of distributed and microservice architecture in the orderly charging system, and describes in detail the technical architecture, design ideas and specific implementation of the system. It can provide a strong guarantee for the large-scale application of charging piles and orderly charging, promote the orderly charging of users, and more effectively popularize the power demand response on the user side.

Acknowledgment
Thank you for your participation in this project. It's your hard work that makes us such an excellent software product. Thank my colleagues in the team for their help in writing this paper and gathering materials.

References
[1] Shugang Yin, Taorong Gong, Rui Liu etc. Design and application of intelligent charging system of electric vehicle based on cloud platform[J]. Distribution & Utilization, 2015(7), pp.43-47.
[2] En Ni. High concurrency practice of Netty、Redis、ZooKeeper. China Machine Press,2019,337-353.
[3] Chao Ni. Principle and practice of distributed consistency from Paxos to zookeeper. Publishing House of Electronics Industry, 2015, 298-309.
[4] Norman Maurer, Marvin Allen Wolfthal. Netty in action. Posts & Telecom Press, 2017, 38-51.
[5] Li Zhou. Microservice architecture practice of Spring Cloud & Docker. Publishing House of Electronics Industry,2017,118-193.
[6] Yong Huang. Architecture Exploration - lightweight microservice architecture (Volume I). Publishing House of Electronics Industry, 2016, 2-4.
[7] John Carnell. Spring Microservices in action. Posts & Telecom Press, 2018,132-163.
[8] Chris Richardson. Microservices Patterns With examples in Java. China Machine Press, 2019, 35-59.
[9] Zhonghua Zhu. Practical Guide of RabbitMQ. Publishing House of Electronics Industry, 2017, 34-54.