Design and Implementation of an Advanced Planning and Scheduling System Based on Microservices

Haoyu Zhang[1,a], Jianming Zhang[3,b], Yaozong Wang[2,c,*]

[1]College of Computer and Cyber Security, Fujian Normal University
[2]Fujian Provincial Key Laboratory of Intelligent Identification and Control of Complex Dynamic System
[3]Quanzhou Institute of Equipment Manufacturing Haixi Institutes, Chinese Academy of Science

[a]e-mail: qsz20211335@student.fjnu.edu.cn
[b]e-mail: zhangjianming@fjirsm.ac.cn
[c]e-mail: *Corresponding auther e-mail:yzwang@fjirsm.ac.cn

Abstract. In response to the problems faced by current Advanced Planning and Scheduling Systems (APS), including their difficulty to scale, deploy, high coupling between modules, and lack of versatility, we have meticulously divided each function of the system based on the original monolithic architecture, constructing an APS based on a microservices architecture. Utilizing Docker containerization technology and the Kubernetes container orchestration engine, we orchestrate and deploy containers loaded with various microservice components, making the system more elastic and flexible. On this basis, with the aid of common services such as Jenkins and the Harbor private repository, we have achieved agile development of the system, enabling the APS system based on the microservices architecture to better adapt to rapidly changing environments and demands. System simulations have shown that the improved system has better agility, scalability, decoupling, and maintainability.

Keywords: microservice; Containerization technology; Advanced planning execution system

1 Introduction

Since the start of the 12th Five-Year Plan, China's manufacturing industry has achieved significant progress, making positive contributions to the transformation and upgrading of the Chinese economy and the advancement of global manufacturing. However, with the continuous advancement of computer networking technology and the progress of manufacturing informatization, manufacturing enterprises are facing increasingly fierce domestic competition and international challenges. To enhance production capacity and management level, the manufacturing industry needs to implement enterprise management systems. Currently, most domestic enterprises use ERP (Enterprise Resource Planning) systems, which encompass multiple functional modules such as

A. Rauf et al. (eds.), *Proceedings of the 3rd International Conference on Management Science and Software Engineering (ICMSSE 2023)*, Atlantis Highlights in Engineering 20,
https://doi.org/10.2991/978-94-6463-262-0_61

finance, procurement, sales, inventory, production, and human resources. The aim is to improve operational efficiency and decision-making capabilities. However, these systems lack flexibility and real-time capabilities in production planning and scheduling, failing to adapt to today's dynamic market environment. Therefore, it is necessary for enterprises to deploy APS (Advanced Planning and Scheduling) systems to compensate for the deficiencies in production planning and scheduling. This will improve the operational efficiency of production lines, enhance the flexibility and real-time nature of planning, and ultimately enhance the competitiveness of the enterprise.

Due to the complex and volatile market environment, the number of orders handled by enterprises frequently changes. Traditional monolithic architectures of Advanced Planning and Scheduling systems, due to high coupling between components, are unable to effectively scale up or down according to order volume. This presents a series of problems for enterprise development and subsequent maintenance. For instance, during a sudden increase in order volume in a particular quarter, a monolithic APS system cannot handle such a large number of orders. Rebuilding an entire system is time-consuming, and the inability to provide reasonable production schedule arrangements within a short time greatly affects the progress of the enterprise. Therefore, enterprises require a flexible and scalable APS system based on a microservices architecture to address the limitations of the existing system and support future development. Such an advanced planning and scheduling system can better adapt to changes in market demand and facilitate effective production line planning and resource allocation. By collecting and analyzing real-time data, enterprises can make more accurate decisions, improve production efficiency, and promptly respond to fluctuations in order volume.

Microservice architecture is a distributed system architecture emerging in recent years. It was proposed by Martin Fowler and James Lewis in 2014. Its predecessor is service-oriented Architecture (SOA)[1]. but SOA has some shortcomings such as too complex, difficult deployment and low performance. The microservices architecture pattern solves the SOA architecture problem by breaking up a large application into smaller services. Microservices emphasize distribution and decentralization, dividing services at a finer granularity, each running in its own process, and communicating using lightweight communication mechanisms such as REST API[2]. Key features of the microservices technology architecture include: loose coupling, expandability, fault tolerance, multilanguage support, autonomy, etc.

The current system is deployed in a server cluster using containerization technologies such as Docker and Kubernetes (abbreviated as K8s). Containerization is a lightweight form of virtualization that allows applications and their dependencies to be packaged into separate, isolated runtime environments. Docker is the most popular containerization engine, providing a complete ecosystem of components including images, containers, and repositories. Kubernetes, an open-source container orchestration platform developed by Google, offers automated deployment, scaling, and management functionalities for container clusters. With the support of these containerization technologies, the microservices-based APS system can effectively scale up or down, as well as achieve load balancing and other capabilities[3].

The subject of this paper is a domestic clothing accessories manufacturing company. Due to the nature of its business, the company faces frequent changes in requirements

and evolving customer demands, necessitating timely adjustments to adapt to the market. Therefore, to effectively respond to the rapidly changing environment and requirements, and to enhance product efficiency and quality, the APS system for this enterprise needs to implement agile development. Agile development is a software development methodology based on iteration, adaptability, and collaboration. Through agile development, the development team can flexibly adjust priorities and tasks, promptly respond to new requirements, and ensure that the project meets customer expectations[4].

2 Design and Implementation

2.1 Demand Analysis

Through field visits and problem analysis of the production workshop, it has been determined that the APS system for the workshop should have the following functional modules: System Dashboard, Workshop Management, Production Management, Production Planning Management, Statistical Charts, and Visualization Dashboard. Among them, the System Dashboard module includes features such as login and registration. The Workshop Management module is responsible for monitoring the operational status of equipment, material inventory, and the presence of personnel at each workstation. The Production Management module primarily handles production scheduling and management. The Production Planning module subdivides plans into master plans and sub-plans based on specific scheduling arrangements, which are then disseminated to various workshops and positions. The Statistical Charts module calculates the current workshop's production progress and the overall order progress based on the reported work of workshop personnel[5]. The Visualization Dashboard module displays comprehensive data, including workshop progress, order progress, work reporting, material usage, and quarterly performance. The specific business modules are illustrated in Figure 1.

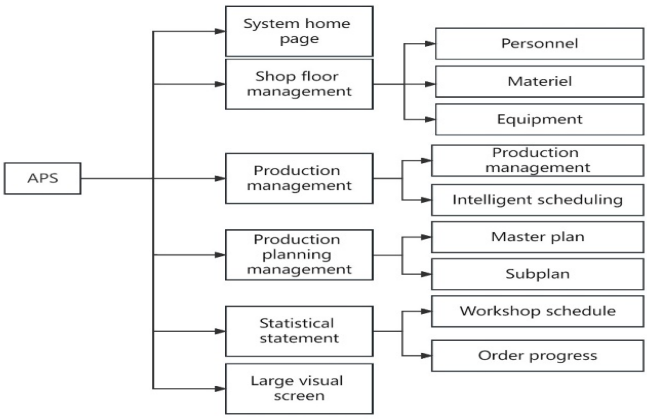

Fig. 1. APS service module diagram

2.2 Business Unbundling and Architecture Implementation

Based on the requirements analysis and the aforementioned functional modules, combined with the actual production process, it can be concluded that there is frequent data exchange between the production management and production planning management modules. Therefore, there is a strong coupling between these two sets of functionalities, and they need to be merged into the same production management microservice. Due to the significant resource consumption involved in order scheduling, the intelligent scheduling component of the production management module will be separated and developed as a separate microservice called the scheduling engine. To centralize the management of security-related functionalities such as password encryption, identity verification, and access control, in order to enhance system security and protect user privacy, a login and registration microservice will be set up. Therefore, the improved system primarily consists of the following microservices: login and registration microservice, workshop management microservice, production management microservice, scheduling engine microservice, statistical reporting microservice, and visualization dashboard microservice.

These microservices are packaged into Docker images and deployed on a cluster of multiple servers using Kubernetes. Each microservice returns its response results to the microservice gateway via API interfaces, enabling information exchange between microservices as well as between the frontend and backend. This microservice design is based on Node.js and the Loopback 4 framework.

To facilitate the exchange of information between microservices, a service discovery and registration mechanism is needed. Service discovery in the context of microservices architecture refers to the ability of services to automatically discover and locate each other[6]. Manually managing a large number of service instances can be challenging, so an automated mechanism is required for service discovery. Common service discovery mechanisms include DNS-based service discovery, client-side load balancing, and service registries. Through service discovery mechanisms, services can quickly find the other services they depend on, enabling efficient communication between services. The Service mechanism in Kubernetes aligns well with this requirement. It abstracts the different IP addresses of pods hosting multiple microservices into a publicly accessible network and provides them with the same DNS, allowing communication between multiple pods and enabling service discovery and load balancing, among other features.

In large-scale distributed systems, there is frequent interaction between services, and manually tracking service calls can be highly inconvenient[6]. To achieve goals such as fault localization and troubleshooting, performance optimization, service topology visualization, SLA monitoring[7], and compliance checks, the revamped APS system needs to implement a service tracing mechanism. In the K8s cluster, deploying Zipkin can enable service tracing and monitoring.

In a microservices architecture, where each microservice runs independently, effective logging is crucial for troubleshooting and issue resolution when the system experiences failures, errors, or exceptions[8]. By recording detailed log information, developers can quickly identify the service and request involved in the problem and pinpoint the specific source of the error. Log recording not only provides traceability for system

errors but also proves valuable in performance optimization and capacity planning. By analyzing log data, system bottlenecks and performance issues can be identified, leading to appropriate adjustments and optimizations[9]. Additionally, collecting and analyzing logs provides deeper insights into business activities. For example, monitoring user behavior, tracking business processes, and analyzing usage patterns can all be achieved through log analysis[10].In the APS system, there are two client interfaces: a web application and a mobile app. According to the above description, the improved APS system architecture can be shown in Figure 2.

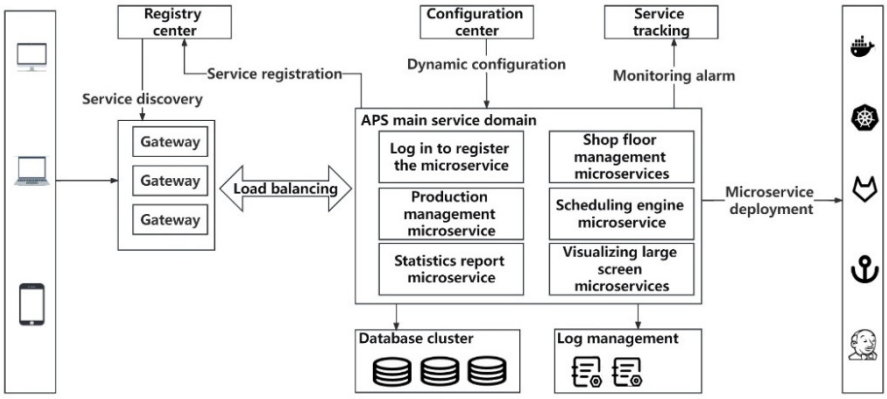

Fig. 2. Overall architecture diagram of APS system

The aforementioned APS system based on the microservices architecture will be deployed in a K8s cluster. The cluster used in this experiment comprises five servers, two of which serve as master nodes, while the remaining three operate as worker nodes. The scale of the cluster is shown in Figure 3.

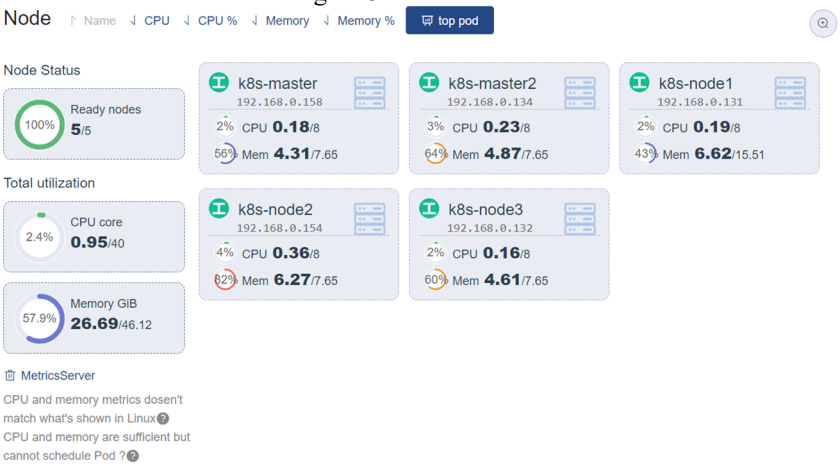

Fig. 3. K8s Cluster scale

In order to facilitate the scalability of the APS system and achieve isolation between microservices, the core business parts of the system are placed in different pods, as shown in Figure 4. When business requirements change, it is very straightforward to make adjustments to different microservices.

```
root@k8s-master:~# kubectl get pods -n aps-dev
NAME                                    READY   STATUS    RESTARTS      AGE
iplannerv5-deployment-84c64858b7-sbmfb  1/1     Running   2 (14d ago)   87d
loginserverce-6f6bc8fc5f-hdzrd          1/1     Running   0             21h
loginserverce-6f6bc8fc5f-pkzq6          1/1     Running   0             21h
loopback-deployment-6fbc796cdd-5slds    1/1     Running   2 (14d ago)   86d
nginx-deployment-58df48c4bb-xswrq       1/1     Running   1 (14d ago)   62d
postgres-deployment-66f6c44bf9-d7mqg    1/1     Running   3 (14d ago)   90d
productionservice-6f6bc8fc5f-8l65p      1/1     Running   0             21h
productionservice-6f6bc8fc5f-vsvpn      1/1     Running   0             21h
report-6f6bc8fc5f-m7cf5                 1/1     Running   0             21h
shopservice-6f6bc8fc5f-pdxph            1/1     Running   0             21h
shopservice-6f6bc8fc5f-srtbv            1/1     Running   0             21h
web-6f6bc8fc5f-4bn8v                    1/1     Running   0             21h
web-6f6bc8fc5f-hxx2x                    1/1     Running   0             21h
web-6f6bc8fc5f-jcct4                    1/1     Running   0             21h
zipkin-6f6bc8fc5f-mwf9v                 1/1     Running   0             21h
root@k8s-master:~#
```

Fig. 4. APS core service pods

2.3 Improvement effect

In order to verify the advantages of APS system based on microservice architecture over traditional APS architecture, we conducted a comparative simulation experiment, and the results are shown in Table 1.

Table 1. Comparison of simulation results

Operation	Concurrent volume	Max response time		Average response time	
		single-service	micro-service	single-service	micro-service
Order entry	50	1.021s	0.502s	0.651s	0.322s
	100	1.765s	0.695s	1.132s	0.585s
	200	2.325s	1.024s	1.952s	0.883s
Report query	50	1.126s	0.601s	0.692s	0.352s
	100	1.835s	0.695s	1.253s	0.593s
	200	2.525s	1.138s	2.012s	0.903s
Shop monitoring	50	1.009s	0.402s	0.643s	0.301s
	100	1.685s	0.595s	1.112s	0.512s
	200	2.257s	0.924s	1.812s	0.872s

The testing machine used for this experiment has a Windows 10 operating system, an AMD Ryzen 5 processor, and 8GB of memory. The stress test tool used in this experiment is Apache JMeter. The experimental data was generated from the production of auxiliary clothing materials, and a web crawler program was used to simulate

browser access to the server. We selected several commonly used system operations and counted the response time of different systems through crawler simulation of different visits. As can be seen from the above table, microservices architecture has significant advantages in handling multiple concurrency.

2.4 Implementation of agile development

The agile implementation of this system is based on the Jenkins automation tool. We have set up Jenkins, GitLab code repository, and Harbor private image repository as common services on the K8s cluster. GitLab code repository and Harbor container image repository are interconnected through Jenkins. Jenkins triggers continuous integration and deployment by deploying a webhook on GitLab. When developers perform actions such as code submission or code merging, Jenkins pulls the code from the code repository and performs operations such as compilation, packaging, and testing. Once the build and tests are successful, Jenkins utilizes its built-in Docker plugin to publish the packaged image to the private image repository. The Jenkins Kubernetes plugin is then used to pull the new version of the image from the private repository and deploy it within the cluster, enabling agile development[4]. The process is illustrated in Figure 5.

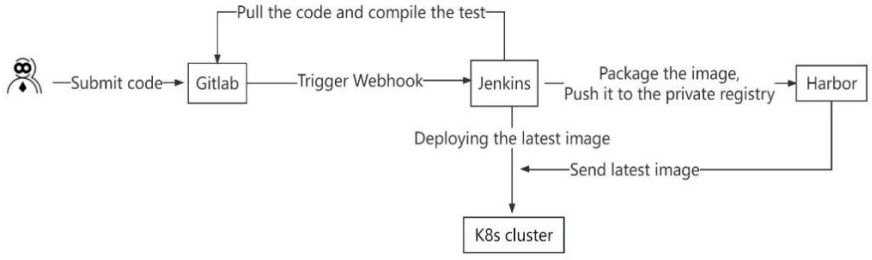

Fig. 5. Agile development implementation process

3 Conclusions

This paper presents the improvement and design of the APS system based on microservices architecture. The enhanced system demonstrates higher agility, robustness, and scalability, providing a strong solution to tackle the challenges posed by frequent project updates and explosive data growth. Additionally, agile development practices were proposed and implemented for the improved system, enabling rapid adjustments according to enterprise needs. The enhancement approach for the microservices-based APS holds significant importance in today's business environment. The experiences and insights gained from this study are valuable for the design and optimization of similar systems, offering beneficial references for enterprises to maintain a competitive edge in the era of digital transformation.

Acknowledgment

The authors recognize the support of this work by the STS program of the Fujian Science and Technology Sciences Project under Grant(2022T3025).

References

1. Niknejad N, Ismail W, Ghani I, et al. Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation[J]. Information Systems, 2020, 91: 101491. https://doi.org/10.1016/j.is.2020.101491
2. Mantravadi S, Srai J S, Brunoe T D, et al. Exploring reconfigurability in manufacturing through IIoT connected MES/MOM[C]//2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). IEEE, 2020: 161-165. [doi: 10.1109/IEEM45057.2020.9309989]
3. Gias A U, Casale G, Woodside M. ATOM: Model-driven autoscaling for microservices[C]//2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019: 1994-2004. [doi: 10.1109/ICDCS.2019.00197]
4. Balalaie A, Heydarnoori A, Jamshidi P. Microservices architecture enables devops: Migration to a cloud-native architecture[J]. IEEE Software, 2016, 33(3): 42-52. [doi: 10.1109/MS.2016.64]
5. Xu S S D, Chang T C. A feasible architecture for ARM-based microserver systems considering energy efficiency[J]. IEEE Access, 2017, 5: 4611-4620. [doi: 10.1109/ACCESS.2017.2657658]
6. Vural H, Koyuncu M. Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice? [J]. IEEE Access, 2021, 9: 32721-32733. [doi: 10.1109/ACCESS.2021.3060895]
7. Baarzi A F, Kesidis G. Showar: Right-sizing and efficient scheduling of microservices[C]//Proceedings of the ACM Symposium on Cloud Computing. 2021: 427-441. [doi: 10.1145/3472883.3486999]
8. Soldani J, Brogi A. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey[J]. ACM Computing Surveys (CSUR), 2022, 55(3): 1-39. [doi: 10.1145/3501297]
9. Tak B C, Tao S, Yang L, et al. Logan: Problem diagnosis in the cloud using log-based reference models[C]//2016 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2016: 62-67. [doi: 10.1109/IC2E.2016.12]
10. Roelke R. Dynamic Causal Monitoring for Distributed Systems[J]. ACM Trans. On Computer Systems, 2017, 35(4): 11. [doi: 10.1145/3208104]