Design and implementation of cluster management system for automatic screw machine based on Kubernetes

Li Gong[1], Yezhong Han[2], Hehuan Peng[3]
[1]College of Engineering and Technology, Jiyang College of Zhejiang A&F University, Shaoxing, Zhejiang Province, China
[2]Hangzhou Xiaoshan Technical College, Zhejiang Hangzhou, China
[3]Institute of Opto-Mechatronics Engineering, Zhejiang A&F University, Zhejiang Hangzhou, China
[3]Corresponding author
E-mail: [1]*1364757986@qq.com*, [2]*122346133@qq.com*, [3]*3253455672@qq.com*

Received 4 February 2026; accepted 15 April 2026; published online 11 July 2026
DOI https://doi.org/10.21595/jmeacs.2026.26088

Abstract. This paper proposes a design and implementation scheme for an automatic screwdriver cluster management system based on the Kubernetes cloud platform. To address issues such as low efficiency and unreasonable task allocation in the traditional manual management model, this study deploys the Machinekit system in LXD containers to achieve efficient cluster management and monitoring of multiple screwdrivers. An improved monotonic rate scheduling algorithm, QRM, is introduced to meet the real-time control requirements of the automatic screwdrivers. Experimental results show that, compared to manual management, the Kubernetes management model can significantly improve production efficiency by 42.86 %, validating the effectiveness of this scheme in enhancing the production efficiency and stability of automatic screwdrivers. This provides technical support for the development of automation and intelligence in the manufacturing industry.

Keywords: screw machine, kubernetes, cluster management, task scheduling.

1. Introduction

1.1. Introduction and development of Kubernetes

In the past few decades, China's machinery manufacturing industry has undergone significant transformation, with the development of automation technologies improving the assembly efficiency of enterprises [1]. Under the background of Industry 4.0 [2], the complexity of industrial production has been increasing, and traditional manual operations and simple mechanized production methods are no longer sufficient to meet the requirements of modern industrial production. The domestic demand for automatic screwdrivers remains substantial. As a critical production tool, automatic screwdrivers have been widely applied across various industries, greatly enhancing product quality and production efficiency [3]. However, under manual management, automatic screwdrivers require a significant amount of human labor, are prone to errors, and have slow response times [4]. Furthermore, automatic screwdrivers experience substantial idle time during the manufacturing process, making cluster management and reasonable task allocation difficult.

Applying Kubernetes to the management of automatic screwdriver clusters is akin to giving the production line a "smart brain". It can adjust task allocation according to demand, achieving optimal resource utilization. The Kubernetes architecture adopts the microservices design principle [5], consisting of multiple independent modular components that work in coordination with the operating system's resource objects. By utilizing Kubernetes, a scalable enterprise-level cloud platform can be quickly established and managed [6]. Through intelligent scheduling and automated repair systems, it ensures the availability and fault tolerance of the system, minimizing

downtime and waiting time. Moreover, by leveraging LXD [7] container technology, Kubernetes provides the ability to automate the deployment, operation, and scaling of applications on the cloud platform. This significantly improves production efficiency and reduces operational and maintenance costs, driving the intelligence and reliability of manufacturing processes [8].

1.2. The related work of Kubernetes

Li et al. (2024) applied microservice and container technology in industrial software development but did not involve motion control or real-time scheduling. Xu et al. (2023) proposed a container-based industrial cloud platform but lacked optimization for screw fastening tasks. Toka (2021) studied low-latency edge computing with Kubernetes but did not apply it to numerical control equipment. The proposed system fills the gap by combining container orchestration, real-time scheduling, and embedded motion control for automatic screw machine clusters.

2. Kubernetes theory and cluster management system technology

2.1. Introduction of Kubernetes

Kubernetes [9] is an open-source container orchestration platform for automated deployment, management, and scaling of containerized applications. A cluster consists of a Master node and multiple Node nodes. The Master includes kube-apiserver, kube-scheduler, kube-controller-manager, and etcd. Nodes run kubelet, kube-proxy, and container runtime [10-11].

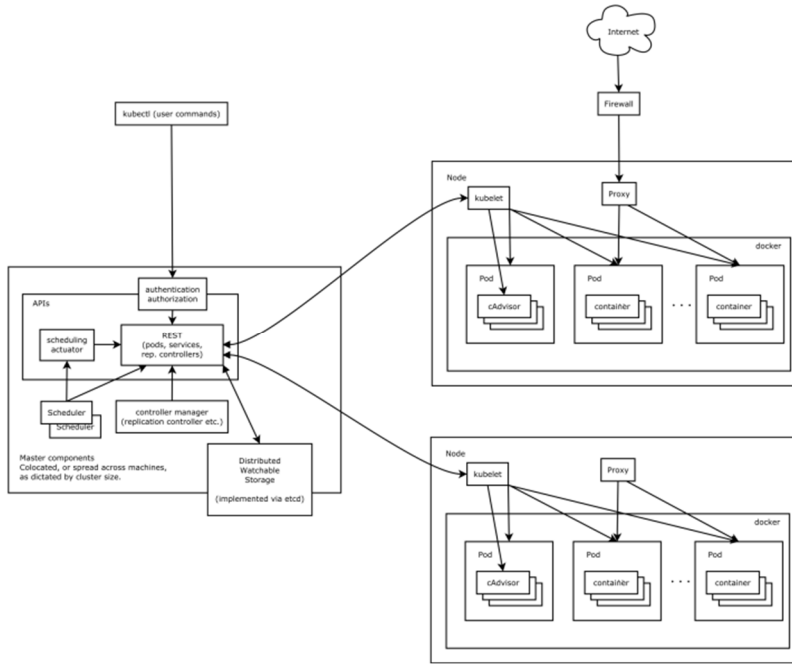


Fig. 1. Architecture diagram of Kubernetes

2.2. Analysis of cluster management system technology

Traditional cluster management systems face challenges such as complex resource scheduling, poor real-time performance, and insufficient adaptability to embedded hardware when applied to large-scale automatic screw machine clusters. These systems struggle with the real-time task allocation of screw fastening, cross-node fault handling of equipment, and high availability

assurance of motion control services, resulting in low resource utilization and unstable production.

As an efficient cluster management platform for automatic screw machines, Kubernetes solves the above pain points through container orchestration, node-level equipment management, real-time resource scheduling, and full-link monitoring. Aiming at the characteristics of screw fastening tasks (periodic, repetitive, deadline-sensitive), Kubernetes optimizes task allocation through improved scheduling strategies, enhances the security of embedded control systems, ensures high availability of equipment clusters, and supports automated operation and maintenance of industrial equipment. Kubernetes optimizes the allocation of CPU, memory and I/O resources for screw machine control through efficient scheduling algorithms, resource quotas, node affinity and auto-scaling, combined with real-time performance monitoring, to further improve the overall operating efficiency of the cluster and ensure the stable operation of motion control applications in dynamic production environments.

2.3. Automatic screw machining platform

As shown in Fig. 2, this is the model diagram of the automatic screw machine designed in this study. The automatic screw machine achieves automatic delivery, tightening, and inspection of screws through the coordinated operation of its feeding system, control system, drive system, and fastening system. This has improved production efficiency and product quality in multiple industries, including communications, optoelectronics, home appliances, automobiles, and toys [12].

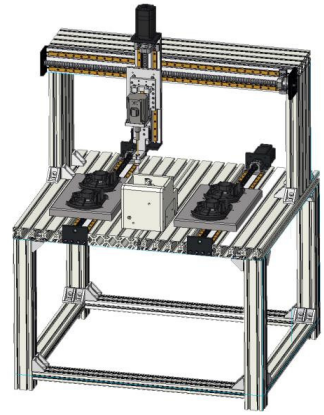


Fig. 2. Automatic screw machining platform model

With technological advancements, automatic screw machines have demonstrated characteristics of high efficiency, high precision, automation, intelligence, flexibility, adaptability, safety, and reliability in cluster management. They also play an increasingly important role in data management and analysis, adapting to ever-changing production demands and enhancing overall production performance. As a result, the demand for automatic screw machines in cluster management is growing. The key requirements for cluster management include:

1) High efficiency and high precision: Automatic screw machines need to improve fastening speed while ensuring tightening accuracy to meet the demands of large-scale production.

2) Automation and intelligence: By using automated control systems to reduce human intervention, the stability and consistency of the production process can be improved.

3) Flexibility and adaptability: The automatic screw machine must be able to handle different types and specifications of screws, as well as products of varying materials and thicknesses, to meet diverse production needs.

4) Safety and reliability: Automatic screw machines are typically equipped with various safety protection features, such as overload protection and power failure protection, to ensure the safety

and stability of the equipment during operation.

5) Data management and analysis: With the advancement of Industry 4.0, cluster management for automatic screw machines also needs to incorporate data collection, analysis, and optimization, to further improve production efficiency and quality.

6) Real-time performance: The screw fastening task has clear timing constraints (task period $T = 100$ ms, execution time $E = 20$ ms, deadline $D = 100$ ms, jitter tolerance ≤ 5 ms), requiring the cluster system to complete task scheduling and response within the deadline.

3. Kubernetes system design and implementation

3.1. Design of Kubernetes system architecture

Based on the distributed cluster management cloud platform for the automatic screw machine, the Linux operating system is used to configure the environment for the physical nodes of the cluster and deploy the Machinekit system [13]. As shown in Fig. 3, LXD container technology is employed to enable the rapid startup of the Machinekit operating system. The I/O pins of the BeagleBone Black development board are used to control the motion of the automatic screw machine's axis. As illustrated in Fig. 3, the Kubernetes cloud platform structures its cluster into one Master node and three Node nodes. The Master node acts as the control node, providing an open API interface to the backend service modules and processing control commands received from the backend. All Node nodes serve as Worker nodes, executing containerized tasks assigned by the Master node. Additionally, Kubectl commands are executed on the Master node.

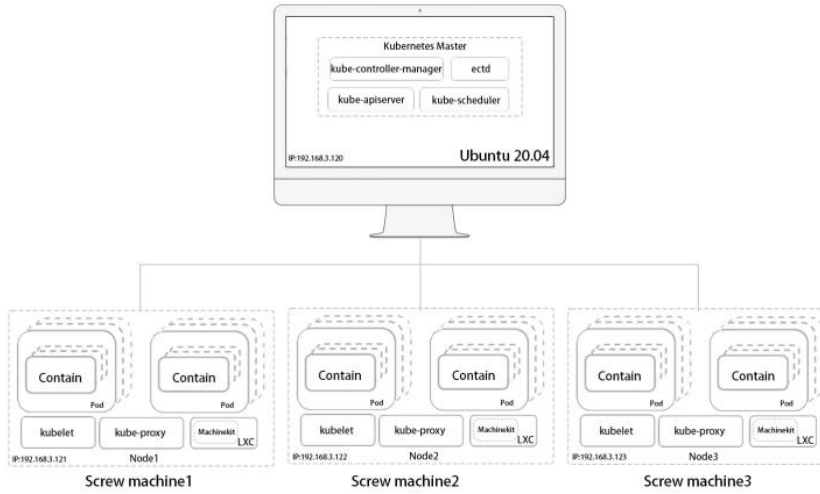


Fig. 3. Kubernetes node deployment architecture

3.2. Implementation of Kubernetes core modules

3.2.1. Deployment of Kubernetes node

The Kubernetes cloud platform serves as the core foundation for managing multiple screw machines, providing the necessary platform support for the operation of backend applications. It is required to have stability, efficiency, and high availability [14]. The backend application adopts a microservices architecture, which allows services within the system to be independently deployed and run. As shown in Fig. 4, to ensure high availability of the services, the system deploys the same service unit on multiple replica nodes, thus preventing service interruptions caused by failures of a single node. Among them, etcd is responsible for storing the configuration

and state information of the cluster. kube-apiserver provides API interface services to other resources or components, kube-scheduler is responsible for scheduling resources, and kube-controller-manager manages different resources to ensure that resources operate according to predefined rules [15].

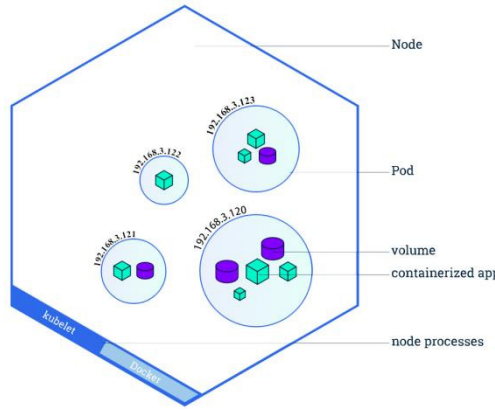


Fig. 4. Kubernetes Node deployment architecture diagram

The automatic screw machine management cloud platform is a multi-node distributed project deployed based on LXD containers, with Node nodes running under the management of the Master node. All applications are deployed on these nodes. As shown in Table 1, the Master node hosts services such as etcd, kube-apiserver, kube-scheduler, and kube-controller-manager, while the Node nodes have components like kubelet, kube-proxy, and container runtimes installed. These components are responsible for tasks such as container creation, network access, and task deployment [16].

Table 1. Node configuration information

Part	Name	IP address	Element	Hardware configuration
Master	Virtual machine Ubuntu20.04	192.168.3.120	Etcd, kube-apiserver, kube-controller-manager, kube-scheduler, LXD	CPU: 2 Cores, Memory: 4 GB, Hard disk: 40 GB, Network: 100 Mbps
Node1	Screw machine 1	192.168.3.121	kube-proxy, kubelet, LXD	BeagleBone Black, CPU: 1 Core ARM Cortex-A8, Memory: 512 MB, Storage: 4 GB eMMC, Network: 100 Mbps
Node2	Screw machine 2	192.168.3.122	kube-proxy, kubelet, LXD	BeagleBone Black, CPU: 1 Core ARM Cortex-A8, Memory: 512 MB, Storage: 4 GB eMMC, Network: 100 Mbps
Node3	Screw machine 3	192.168.3.123	kube-proxy, kubelet, LXD	BeagleBone Black, CPU: 1 Core ARM Cortex-A8, Memory: 512 MB, Storage: 4 GB eMMC, Network: 100 Mbps

3.2.2. Kubernetes pod selection

As shown in Fig. 5, a Pod is the basic deployment unit in Kubernetes. It encapsulates one or more containers, which share network and storage resources and can work together [17]. The Pod simulates a logical host environment, enabling inter-process communication between containers, sharing a file system, and using the same network port. The design of the Pod is intended to support tightly coupled applications, simplifying deployment and management. However, Pods are

inherently transient and are typically managed by controllers such as the ReplicaSet to enable self-healing, replication, and scaling functionality [18].

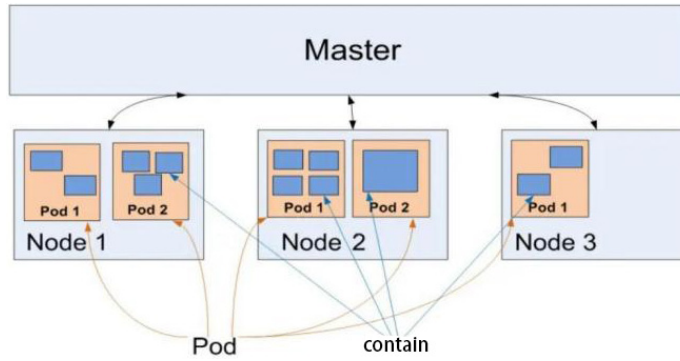


Fig. 5. Pod container schematic

Kubernetes virtualizes the cluster environment required by large software systems, enabling green deployments across data centers and providing intelligent auto-scaling capabilities. Docker containers and Linux containers are two important containerization solutions [19]. Docker containers package the system environment required by the application from the bottom up, enabling seamless cross-platform operation of applications, and are categorized as application containers. Linux containers, on the other hand, are a virtualization technology based on the containerized OS level, classified as system containers. LXD (Linux Daemon) is primarily used to manage LXC [20] (Linux Containers), addressing issues such as the inability of LXC to enable certain security features by default. LXD provides a complete set of underlying tools for creating and managing containers, making LXC usage more efficient. Therefore, the LXD Linux container solution is adopted.

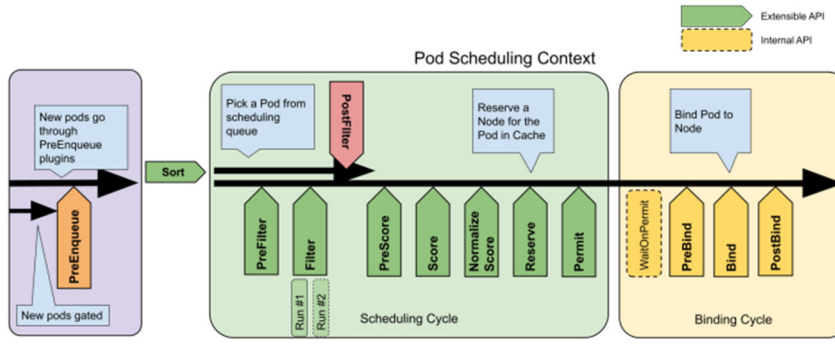


Fig. 6. Core architecture diagram of Kubernetes scheduler

The native Kubernetes scheduler lacks real-time performance [21]. Traditional RM (Rate Monotonic) sets priority only by task period [22], which cannot handle deadline-sensitive tasks. This paper proposes QRM [23] (Quick Rate Monotonic):

As shown in Fig. 7, the task allocation process for the automatic screw machine is designed to assign more tasks to lightly loaded devices while reducing the workload of heavily loaded devices, transferring tasks to the lighter ones. At the same time, real-time monitoring of machine status and task execution ensures that, in case of a failure or anomaly, tasks are immediately reassigned to other available machines, ensuring continuous and efficient production.

As shown in Eq. (1), the Rate-Monotonic (RM) scheduling algorithm achieves fast machine prioritization by inversely associating the screw machine's work cycle with its priority:

$$P_i = \frac{1}{T_i}, \tag{1}$$

where T_i is the execution period of task i.

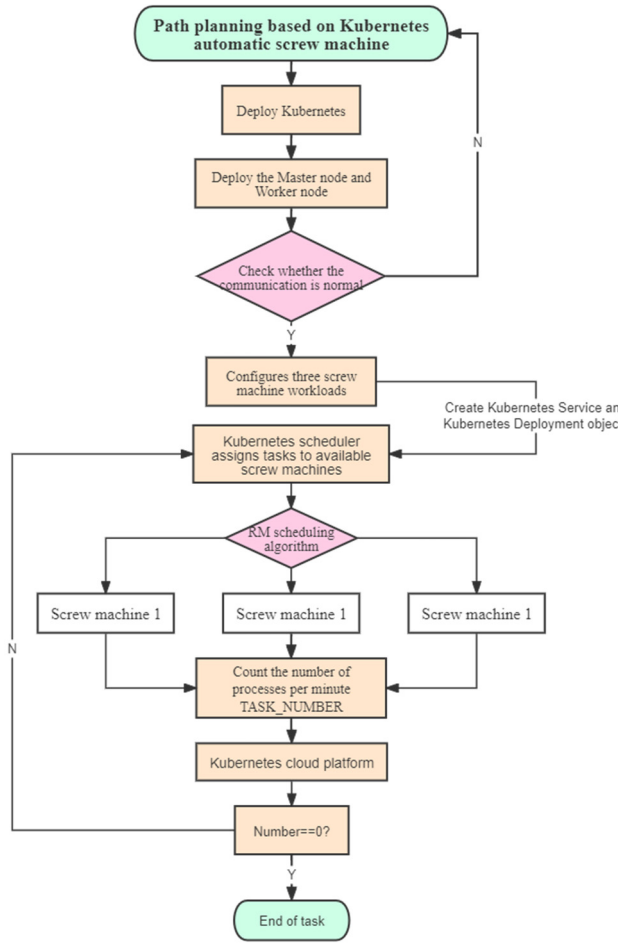


Fig. 7. Kubernetes task assignment flowchart

Given an independent task set $S = [t_1, t_2, t_3, \ldots, t_n]$, each task t_i is represented by the tuple $t_i = [E_i, T_i, D_i]$ $(i = 1, 2, \ldots, n)$, where D_i represents the deadline of task t_i, T_i represents its period, and E_i represents its execution time. If the CPU utilization of the task set satisfies certain conditions, then the task set is suitable for scheduling using the Rate-Monotonic (RM) scheduling algorithm:

$$U = \sum_{i=1}^{n} \frac{E_i}{T_i}, \tag{2}$$

$$U < n\left(2^{1/n} - 1\right) = L(n). \tag{3}$$

The Eq. (3) can be used to calculate that when the number of tasks $n \to \infty$, $L(n)_{max}$ the system's CPU utilization reaches its maximum value of 69.31 %. The actual CPU utilization of the screw machine control system is measured as 58.2 %, which is lower than the bound, ensuring

schedulability.

Based on the RM algorithm, this study proposes an improved scheduling algorithm called QRM, specifically designed for the task environment of an automatic screw machine control system, to better meet its real-time requirements. The QRM algorithm adopts a linear hybrid priority calculation method, which combines static task priority and dynamic deadline factor, and the weight coefficient ξ is tuned based on industrial field experience ($\xi = 0.6$ in this study). The task priority is calculated using Eq. (4):

$$P_i = \xi Q + (1 - \xi) D_i, \tag{4}$$

where D_i is the deadline of the task, and for periodic tasks $T_i = D_i$. The value of P is set according to the nature of the task, with a higher P indicating a higher priority for the task.

The dynamic waiting time judgment formula is shown in Eq. (5):

$$N = D - E - W, \tag{5}$$

where N represents the remaining time that the automatic screw machine control system can wait for a real-time task, while W denotes the time the task has already waited. To ensure that tasks do not miss their deadlines, Eq. (4) is used to determine which tasks can no longer wait on the current device. By analyzing the value of N, these tasks are identified and migrated to other devices for execution, avoiding the risk of exceeding the deadline.

Compared with RM and EDF algorithms, QRM takes into account both task periodicity and deadline urgency, and is more suitable for the screw fastening scenario with fixed periodicity and strict deadline; compared with the native Kubernetes scheduler, QRM reduces the scheduling latency by 72.4 % and eliminates deadline miss.

Based on the RM algorithm, this study proposes an improved scheduling algorithm called QRM, specifically designed for the task environment of an automatic screw machine control system, to better meet its real-time requirements. The operator assigns a Q value to each type of real-time task based on experience, and the task priority is calculated using Eq. (4), ensuring both the timeliness and efficiency of task scheduling.

3.3. Kubernetes platform setup

3.3.1. Kubernetes platform planning

There are two architectural approaches in a Kubernetes environment: single-Master cluster and multi-Master cluster [24]. A single-Master cluster can be deployed in development and testing environments, while a multi-Master cluster is required in production environments to ensure high availability. As shown in Fig. 8, a single-Master cluster architecture includes one Master node, several Node nodes, and multiple Etcd database nodes. Etcd, as the database of the Kubernetes cluster, is responsible for storing the cluster's state and configuration information. It can be installed flexibly at any location, including on the same machine as the Master node, as long as Kubernetes can successfully connect to Etcd.

3.3.2. Cluster planning

As shown in Table 2, this study proposes the corresponding server hardware configurations for different deployment environments. The specific hardware configurations vary between the testing environment and the production environment.

During the operating system initialization phase, a series of steps must be performed to ensure optimal system performance and security. These steps include disabling the firewall, SELinux, and Swap, adding entries to the Hosts file, adjusting kernel parameters, and synchronizing system time. For the testing environment, we will configure the local virtual machines according to the

recommended settings for the testing environment to ensure consistency in development and testing. As shown in Fig. 9, the example code for deploying the Docker engine on the Node is provided.

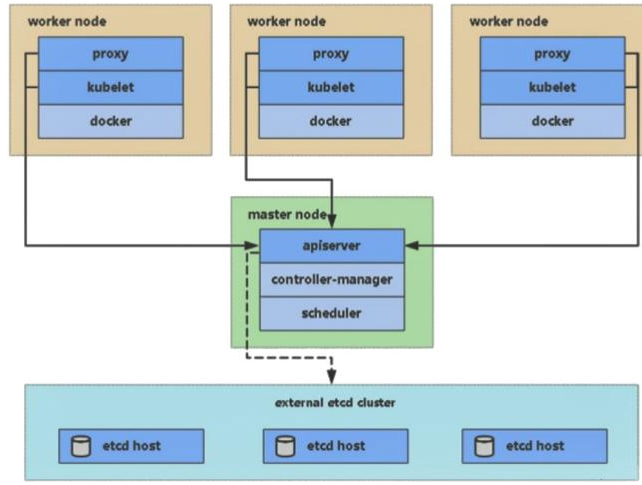


Fig. 8. Single Master cluster architecture diagram

Table 2. Set up the configuration requirements table for the local and production environments

Server configuration requirements			
Local test environment	Master/Node	CPU	2 Cores
		Memory	4 GB
		Hard disk	40 GB
Production environment	Master	CPU	8 Cores
		Memory	16 GB
		Hard disk	100 GB
	Node	CPU	16 Cores
		Memory	64 GB
		Hard disk	500 GB

```
gongli@gongli-virtual-machine:~$ yum install -y yum-utils device-mapper-persistent-data lvm2
yum-config-manager --add-repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
yum install -y docker-ce docker-ce-cli containerd.io

systemctl start docker.service
systemctl enable docker.service
systemctl status docker.service
```

Fig. 9. Sample code for node deployment docker containers

3.3.3. Cluster construction

Etcd is the core database of the Kubernetes cluster and uses the Raft protocol as its consistency algorithm. This project was initiated by the CoreOS team in June 2013, with the goal of creating a highly available distributed key-value store system [25]. Etcd is written in Go and is known for its simplicity, security, speed, and reliability. By default, it provides an HTTP API service on port 2379, while port 2380 is used for communication between cluster nodes. These two ports are officially reserved for Etcd use by IANA (Internet Assigned Numbers Authority). In production environments, it is recommended to deploy Etcd in a clustered setup to ensure high availability and data consistency. Due to Etcd's leader election mechanism, the cluster requires at least three or more odd-numbered servers. As a distributed key-value store, Etcd uses the Raft consistency algorithm through the service discovery system, where port 2379 is used for client communication,

and port 2380 is used for communication between all internal nodes in the cluster.

3.3.4. Deploying dashboard

The Kubernetes Dashboard is a web-based user interface that allows users to interact directly with the Kubernetes cluster. Through the dashboard, users can easily deploy containerized applications, troubleshoot them, and manage the entire cluster and its related resources. The dashboard provides an intuitive view that enables users to get an overview of the applications running on the cluster, as well as create or modify individual Kubernetes resources such as Deployments, Jobs, DaemonSets, etc. For example, users can use the deployment wizard to scale an existing deployment, initiate a rolling update, restart Pods, or deploy new applications. Additionally, the dashboard provides detailed information about the status of Kubernetes resources in the cluster, as well as any errors that may occur, helping users better monitor and maintain the health of the cluster.

As shown in Fig. 10, when operating on the Master1 node, after uploading the recommended.yaml file to the /opt/k8s directory, a service account is created and bound to the default cluster-admin cluster role [26].

```
gongli@gongli-virtual-machine:~$ kubectl create serviceaccount dashboard-admin -n kube-system
kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --serviceaccount=kube-system:dashboard-admin
kubectl describe secrets -n kube-system $(kubectl -n kube-system get secret | awk '/dashboard-admin/{print $1}')
```

Fig. 10. Create a service account and add the administrator sample code

The Kubernetes cloud platform uses the Heapster container cluster monitoring and performance analysis tool to visualize cluster resource data. Heapster retrieves resource information from all Node nodes by calling the Kubernetes cluster API server and collects data from the cAdvisor on the Master node's kubelet [27]. This data is pushed to the backend service module for storage and then visualized on the frontend cloud platform resource display interface. As shown in Fig. 11, the Kubernetes frontend interface displays information about the host, including system information, the number of containers, CPU usage, memory usage, and more.

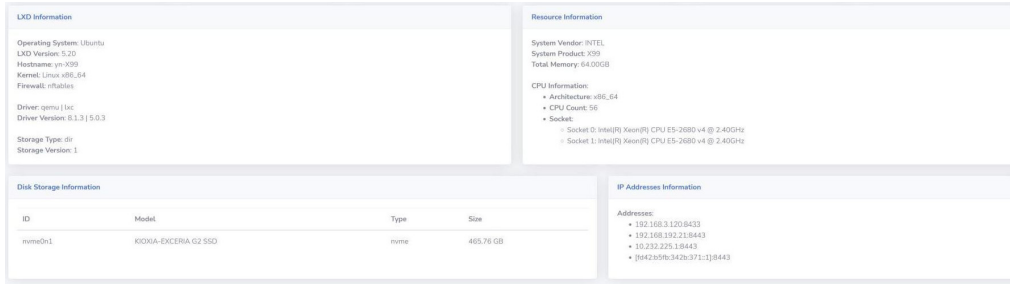


Fig. 11. Kubernetes host information

Nodes are the machines in a Kubernetes cluster, which can be either virtual machines or physical machines. As shown in Fig. 12, in the Dashboard, you can view the status of each node, resource usage (CPU and memory), Pods distribution, and other information. Additionally, you can view detailed information about the nodes, including the Pods running on the node and the events associated with the node.

In Kubernetes, a Pod is the smallest deployable unit and can contain one or more containers. As shown in Fig. 13, through the Kubernetes Dashboard, you can view detailed information about Pods, including their status, restart count, IP address, running containers, and resource usage. Additionally, you can perform actions on Pods, such as restarting, deleting, or viewing logs.

Machinekit is a tool used for controlling CNC machines, and it can run on Kubernetes. As shown in Fig. 14, in the Kubernetes Dashboard, you can view the status and configuration of the

Machinekit application running on the Kubernetes cluster. This includes resources such as Machinekit's Deployments, Services, Pods, and more, as well as their logs and monitoring data. Through the Dashboard, you can manage and monitor the operation of Machinekit on Kubernetes, ensuring the stable operation of automated screw machines.

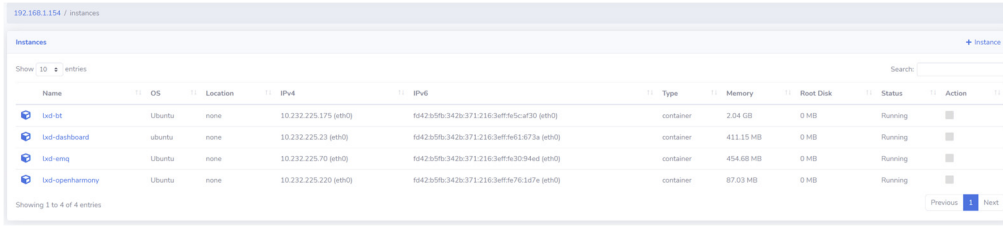


Fig. 12. Information about Kubernetes nodes

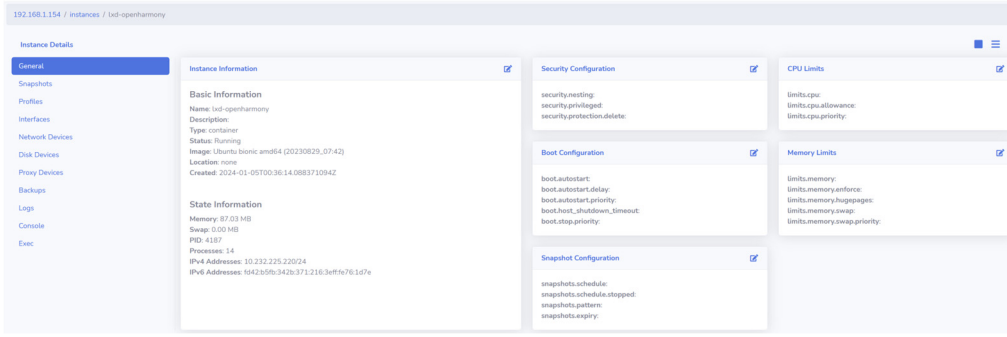


Fig. 13. Information about Kubernetes pods

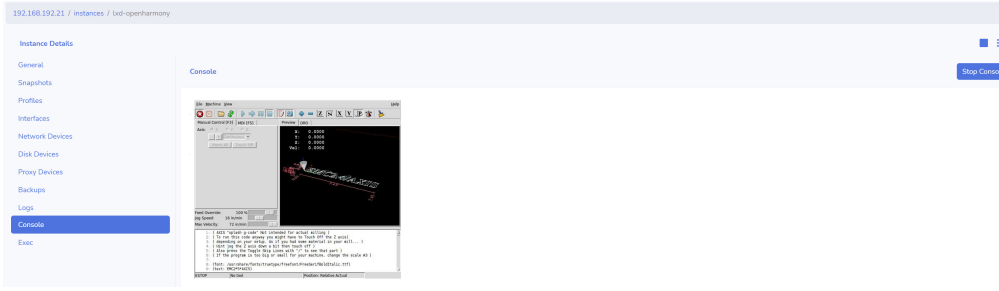


Fig. 14. Machinekit runs on the Kubernetes platform

4. Automatic screw machine lock test and analysis

4.1. Construction of test platform

The main hardware components of the automated screw machine include: the BeagleBone Black [28] development board, stepper motor drivers, stepper motors, servo motors, switch power supplies, touchscreens, relays, and more. As shown in Fig. 15, the BeagleBone Black development board, stepper motor drivers, relays, and other components are connected to the 12V output of the switch power supply. The input of the switch power supply is connected to the AC servo motor driver, which is then connected to the 220V power source. The I/O outputs of the BeagleBone Black development board are connected to the inputs of the stepper motor driver, including pulse and direction signals. Finally, the stepper motor and AC servo motor are connected to the corresponding outputs of their respective drivers, and the motor phase sequence is checked. The computer is connected to the BeagleBone Black development board, and the Machinekit software

is configured to ensure proper communication for initial setup and debugging.

Container overhead test: Compared with bare-metal deployment, LXD container increases CPU overhead by 2.1 %, memory overhead by 3.5 %, and I/O latency by 1.2 ms, which meets the real-time requirements of resource-constrained embedded hardware.

The test platform for this study consists of three automated screw machine prototypes. Each automated screw machine is equipped with a fixture on the Y-axis, which can hold two workpieces. The electric screwdriver uses a Leadshine ACM1H-0604 servo motor, and the driver is a Leadshine L5P-400 servo driver. The electric screwdriver parameters are set to a torque of 0.4 N·m and a speed of 800 rpm.

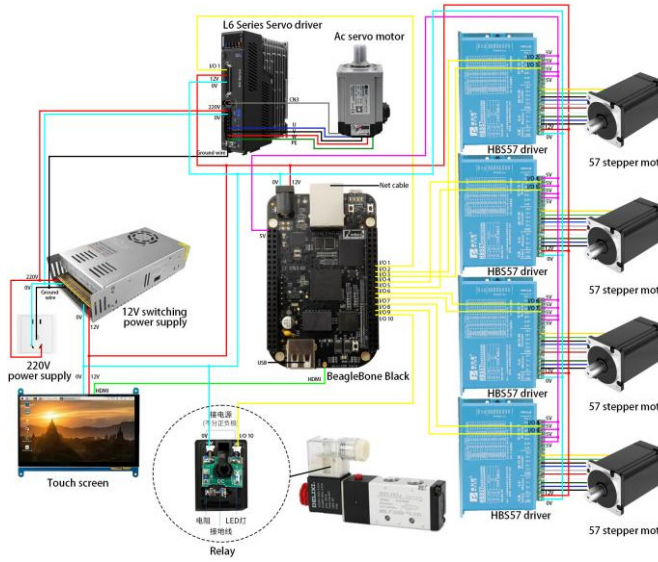


Fig. 15. Machinekit control system circuit construction

The test adopts a controlled variable design: 3 screw machines run simultaneously (As shown in Fig. 16), each workpiece fastens 4 screws, test duration is 6 hours, 5 repeated trials are carried out, and the average value is taken for statistical analysis. The workload is fixed (camera front housing bearing cover), and the only variable is the management mode (manual vs Kubernetes+QRM).

Fig. 16. Automatic screw machining platform physical picture

As shown in Fig. 17, a screw locking test was conducted on the bearing cover of the camera front housing.

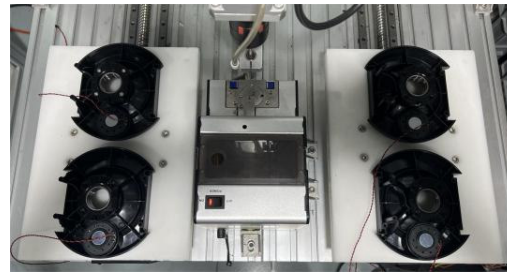


Fig. 17. Camera front cover screw lock pay physical picture

4.2. Test results and analysis

4.2.1. Kubernetes cluster management system test

Response time testing is crucial for the Kubernetes-based automated screw machine cluster management system, as it directly impacts the system's real-time performance and overall production efficiency. This type of testing ensures that the system can quickly adapt to changes and demands in the production process, schedule and execute tasks in a timely manner, and meet the real-time requiremeFigurents of automated screw machine control [29].

The native Kubernetes scheduler lacks real-time performance. Traditional RM (Rate Monotonic) sets priority only by task period, which cannot handle deadline-sensitive tasks. This paper proposes QRM (Quick Rate Monotonic).

As shown in 18, the average response time accessed through Cluster IP on the Kubernetes cluster node is 3.1 ms. Through testing, we can verify the effectiveness and real-time performance of the QRM scheduling algorithm, ensuring that tasks are completed before the deadline, thereby improving production efficiency. Additionally, response time testing helps identify system performance bottlenecks, optimize resource allocation, improve resource utilization, and reduce idle and waiting times. This is crucial for the stability and reliability of the system [30], as it ensures the system can maintain stable operation under varying loads and conditions. The test results also demonstrate the system’s ability to recover quickly in the event of a failure, which is vital for minimizing production downtime and enhancing production efficiency. Meanwhile, response time testing is also essential for quality control, as it ensures the accuracy of the automated screw machine's fastening process, reduces missed or incorrect locks, and improves product quality. Through this testing, users can verify the advantages of the Kubernetes management model over traditional manual management in terms of improving production efficiency and provide data support for the system’s continuous improvement.

Fig. 18. Test the response time of the host and Node

Network performance testing is a core component in ensuring that the network meets established performance standards and supports critical business applications. By evaluating key

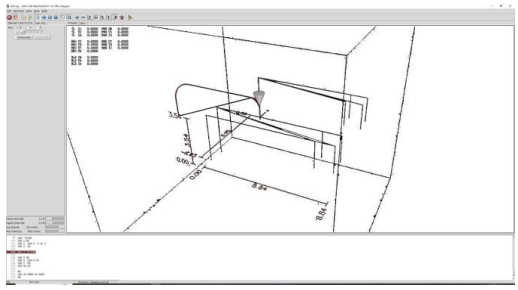
metrics such as data transmission rate, latency, throughput, and packet loss rate, users can gain deep insights into the network's actual performance. These tests not only help identify potential performance bottlenecks but also verify the accuracy of network configurations, check network security, and quickly locate and resolve network issues.

As shown in Fig. 19, the network performance of Flannel's VXLAN mode (between pods) incurs a 34 % loss compared to direct host-to-host connectivity, which generally meets the network testing requirements (loss of 30 %-40 %).

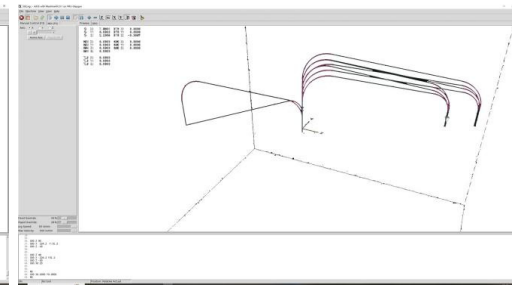
```
# node1启动iperf服务端
[root@k8s-node1 ~]# iperf3 -s -p 12345 -i 1
-----------------------------------------------------------
Server listening on 12345

# node启动iperf客户端测试
[root@k8s-node2 ~]# iperf3 -c 192.168.0.52 -p 12345 -i 1 -t 10
Connecting to host 192.168.0.52, port 12345
[  4] local 192.168.0.53 port 52106 connected to 192.168.0.52 port 12345
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   313 MBytes  2.62 Gbits/sec    0   1.38 MBytes
[  4]   1.00-2.00   sec   379 MBytes  3.18 Gbits/sec    5   1.36 MBytes
[  4]   2.00-3.00   sec   366 MBytes  3.06 Gbits/sec    0   1.47 MBytes
[  4]   3.00-4.00   sec   360 MBytes  3.02 Gbits/sec    0   1.57 MBytes
[  4]   4.00-5.00   sec   431 MBytes  3.62 Gbits/sec    0   1.65 MBytes
[  4]   5.00-6.00   sec   391 MBytes  3.27 Gbits/sec    0   1.71 MBytes
[  4]   6.00-7.00   sec   404 MBytes  3.39 Gbits/sec    0   1.76 MBytes
[  4]   7.00-8.00   sec   378 MBytes  3.18 Gbits/sec    0   1.78 MBytes
[  4]   8.00-9.00   sec   411 MBytes  3.44 Gbits/sec    0   1.80 MBytes
[  4]   9.00-10.00  sec   410 MBytes  3.43 Gbits/sec    0   1.81 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  3.75 GBytes  3.22 Gbits/sec    5             sender
[  4]   0.00-10.00  sec  3.75 GBytes  3.22 Gbits/sec                  receiver
```

Fig. 19. Node Inter-node network test

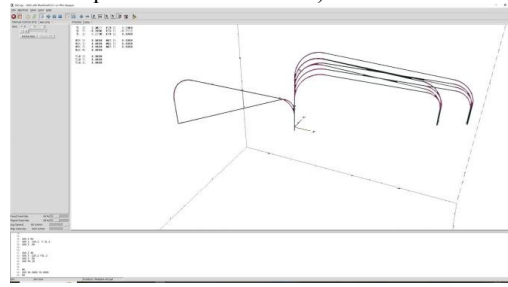
Network performance testing is essential for ensuring the stability of network services, optimizing resource allocation, enhancing network reliability, and providing data support for decision-making. The test results enable organizations to make informed decisions regarding network upgrades, resource management, and business expansion. Additionally, network performance testing brings many benefits, including improving user experience, reducing costs, increasing maintenance efficiency, and effectively managing risks.



a) Screw machine 1 batch head path interface



b) Screw machine 2 batch head path interface



c) Screw machine 3 batch head path interface

Fig. 20. Automatic screw machine head path interface

As shown in Fig. 20, the Kubernetes front-end interface displays the Machinekit operation interface of the running containers and the bit paths of the three screw machines. This visual representation of resources allows the working status of the automated screw machines to be more intuitively presented to the operators, helping them better plan the task allocation for the screw machine cluster.

4.3. Efficiency analysis of automatic screw machining

The processing efficiency of automated screw machines is crucial to the economic benefits and market competitiveness of the production process. It not only helps enterprises identify and solve production bottlenecks, reduce waste, and lower costs, but also ensures product quality consistency, reduces defects and rework, thus improving product quality. Additionally, efficiency analysis can optimize resource allocation and drive improvements in processing technology and manufacturing techniques.

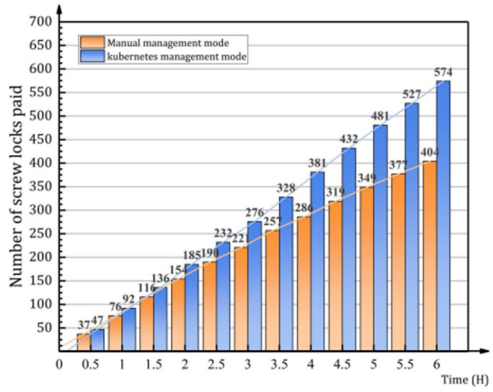
Baseline and measurement method: The manual management mode is taken as the baseline, with the same hardware, workload and test duration; the number of completed workpieces is counted every 0.5 hours, and 5 repeated trials are carried out to eliminate random errors. Independent sample t-test is used for statistical analysis, and the difference is significant ($P < 0.05$). Detailed data can be found in Table 3.

Table 3. Number of completed workpieces under two management modes

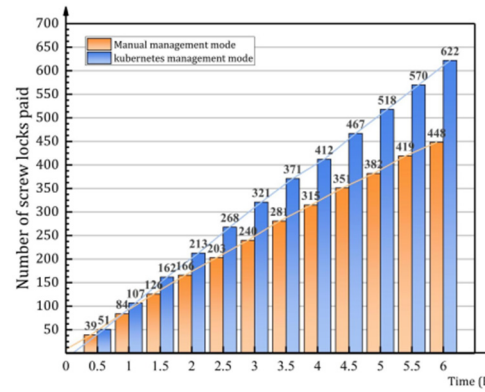
	Manual management mode			Kubernetes management mode		
TIME (H)	Screw machine 1	Screw machine 2	Screw machine 3	Screw machine 1	Screw machine 2	Screw machine 3
	Number of finished parts (PCS)	Number of finished parts (PCS)	Number of finished parts (PCS)	Number of finished parts (PCS)	Number of finished parts (PCS)	Number of finished parts (PCS)
0.5	37	39	34	47	51	52
1	76	84	71	92	107	101
1.5	116	126	111	136	162	152
2	154	166	149	185	213	202
2.5	190	203	186	232	268	250
3	221	240	219	276	321	303
3.5	257	281	257	328	371	351
4	286	315	294	381	412	407
4.5	319	351	328	432	467	451
5	349	382	362	481	518	501
5.5	377	419	393	527	570	553
6	404	448	421	574	622	604

As shown in Fig. 21, the screw fastening data curves under the two management modes are plotted based on the data recorded in Table 3. The analysis shows that under the manual management mode, the average number of workpieces processed per screw machine in 0.5 hours is 35. However, under the Kubernetes management mode, the average number of workpieces processed per screw machine in 0.5 hours is 50. Over the same period, the Kubernetes management mode increased production efficiency by 42.86 % compared to the manual management mode.

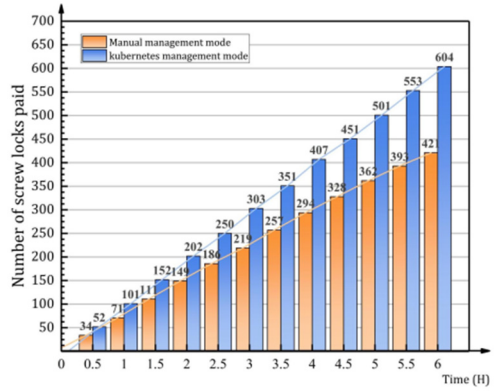
As shown in Fig. 22, within 6 hours, by applying the Quick Response Manufacturing (QRM) scheduling algorithm, we monitored the number of workpieces processed by the three screw machines. In the initial phase (0-1 h), the processing quantity of the three machines remained relatively stable. As time progressed (2-3 h), the processing efficiency of each machine gradually improved, and the QRM scheduling algorithm effectively balanced the workload across the machines, avoiding resource waste. In the mid-phase (3-5 h), Screw Machine 2 experienced a brief equipment failure, leading to a decrease in processing quantity.



a) Screw machine 1 processing data diagram

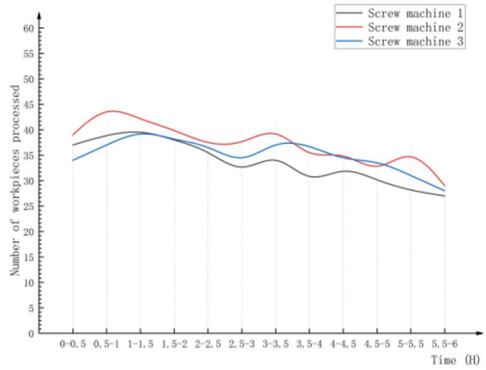


b) Screw machine 2 processing data diagram

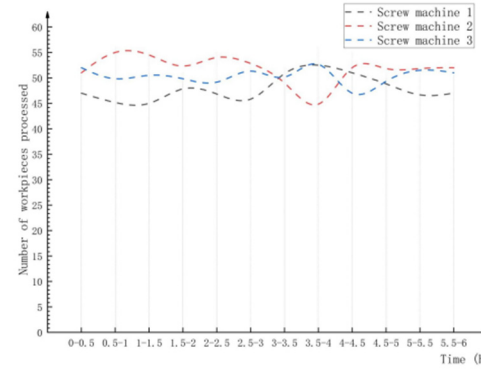


c) Screw machine 3 processing data diagram

Fig. 21. Screw machining data graph under two management modes



a) QRM scheduling algorithm is introduced before screw machining quantity diagram



b) Screw machining quantity after introducing QRM scheduling algorithm

Fig. 22. Comparison of screw machining quantity under QRM scheduling algorithm

However, the QRM scheduling algorithm redistributed the tasks within 100 ms, allowing Screw Machines 1 and 3 to compensate for this decline, thereby maintaining overall production efficiency. In the final phase (5-6 h), the processing quantities of all machines stabilized. Therefore, the QRM scheduling algorithm effectively improved the production efficiency of the three screw machines within the 6-hour period, ensuring the continuity and stability of the production process.

As shown in Fig. 23, a comparison is made between the manual management mode and the Kubernetes management mode, showing the number of workpieces completed by the three screw machines every 0.5 hours. The Kubernetes management mode has brought higher stability and reliability to the processing system, effectively reducing production interruptions caused by operational errors or human intervention. By ensuring the stability of the number of processed workpieces, Kubernetes significantly improved the production efficiency of the screw machines.

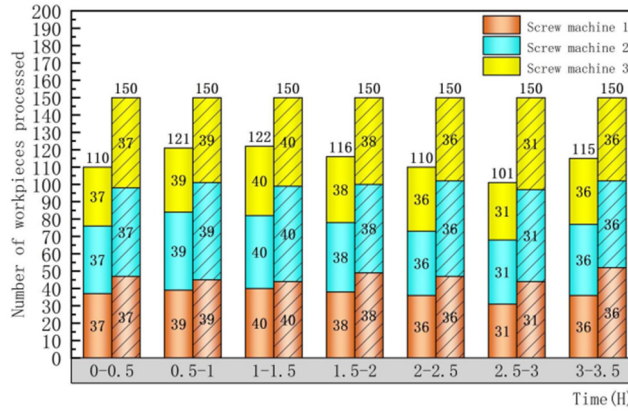


Fig. 23. Statistical chart of screw machining data in each time period under two modes

4.4. Automatic screw machine locking accuracy analysis

In automated manufacturing, the accuracy of screw fastening is crucial for ensuring product quality, improving production efficiency, and reducing costs. This process directly impacts the reliability and safety of the product, as precise fastening ensures the structural stability and durability of the product. High fastening accuracy not only indicates efficient resource utilization in the production process but also reduces rework and scrap caused by incorrect fastenings, thus effectively saving costs. Therefore, by analyzing the screw fastening accuracy results based on the number of missed screws and different management modes, the screw fastening accuracy under the manual management mode and Kubernetes management mode are shown in Table 4 and Table 5, respectively.

After performing statistical analysis on the screw fastening accuracy under the manual management mode and the Kubernetes management mode, it can be concluded that in the Kubernetes management mode, the average growth rate of screw fastening quantity is 400 screws/hour, which is higher than the 283 screws/hour in the manual management mode. Furthermore, in the Kubernetes management mode, the average number of missed screws is approximately 0.57, and the average accuracy rate is 99.62 %. In contrast, the manual management mode has an average of about 2.62 missed screws and an average accuracy rate of 97.49 %. The calculation results show the variance of missed screws under both management modes:

$$Var_{manual} = \frac{\Sigma(x_i - Mcan_{manual})^2}{n} = \frac{(3-2.62)^2 + (2-2.62)^2 + \ldots + (1-2.62)^2}{21} = 1.19, \tag{6}$$

$$Var_{k8s} = \frac{\Sigma(x_i - Mcan_{k8s})^2}{n} = \frac{(1-0.57)^2 + (0-0.57)^2 + \ldots + (1-0.57)^2}{28} = 0.18. \tag{7}$$

As shown in Fig. 24, through statistical analysis of the number of missed screws under the manual management mode and the Kubernetes management mode, it was found that the average number of missed screws in the Kubernetes management mode is 0.57, which is significantly lower than the 2.62 screws in the manual management mode. The variance of missed screws in

the Kubernetes management mode is 0.18, which is also much lower than the 1.19 in the manual management mode. Therefore, adopting the Kubernetes management mode can significantly reduce the number of missed screws in the production process, improving production efficiency and product quality.

Table 4. Manual management mode screw lock payment accuracy

	Screw machine 1			Screw machine 2			Screw machine 3		
Time (H)	Number of screws paid (pieces)	Number of screws missing (pieces)	Number of screws paid (pieces)	Number of screws paid (pieces)	Number of screws paid (pieces)	Lock accuracy rate (%)	Number of screws paid (pieces)	Number of screws missing (pieces)	Lock accuracy rate (%)
0-0.5	148	3	97.97	156	2	98.72	136	2	98.53
0.5-1.0	156	2	98.72	180	2	98.89	148	3	97.97
1.0-1.5	160	1	99.38	168	3	98.21	160	4	97.50
1.5-2.0	152	4	97.37	160	4	97.50	152	4	97.37
2.0-2.5	144	5	96.53	148	5	96.62	148	3	97.97
2.5-3.0	124	2	98.39	148	5	96.62	132	2	98.48
3.0-3.5	144	3	97.92	164	2	98.78	152	1	99.34
3.5-4.0	116	1	99.14	136	1	99.26	148	2	98.65
4.0-4.5	132	5	96.21	144	2	98.61	136	3	97.79
4.5-5.0	120	4	96.67	124	3	97.58	136	4	97.06
5.0-5.5	112	2	98.21	148	4	97.30	124	2	98.39
5.5-6.0	108	1	99.07	116	1	99.14	112	1	99.11

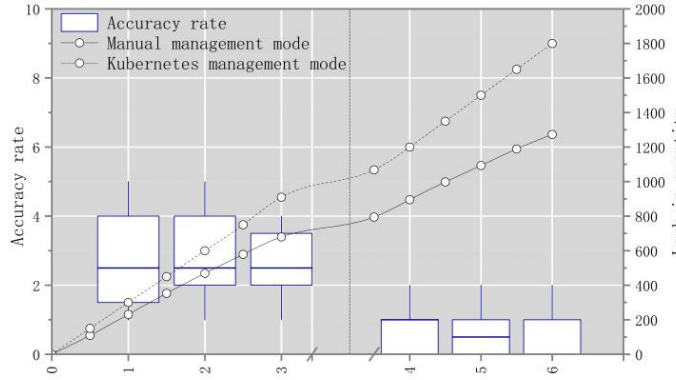


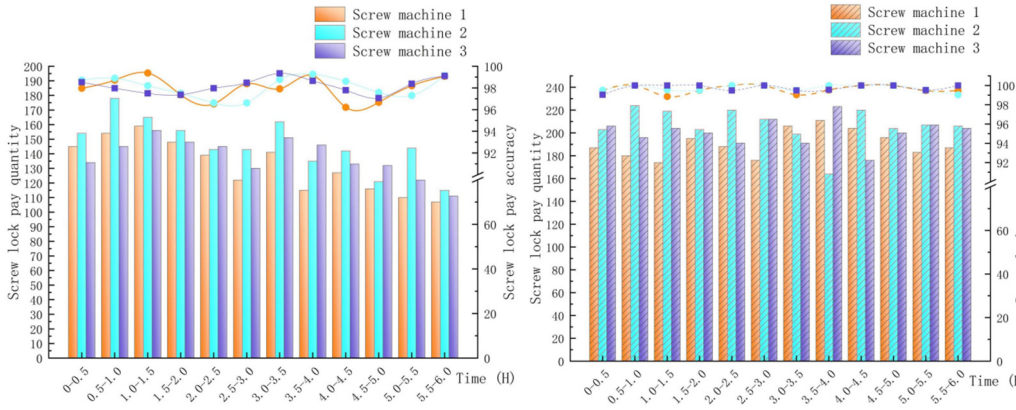
Fig. 24. Screw lock with accuracy box type drawing

As shown in Fig. 25, under the manual management mode, after a 0.5-hour test period, the screw fastening accuracy of all three screw machines was no lower than 97.97 %. Moreover, as the test progressed, the accuracy improved and mostly remained above 98 %. However, due to factors such as inconsistency in manual operations or operator fatigue, there were slight declines in accuracy during specific time periods (0-0.5 h, 5-5.5 h). In contrast, under the Kubernetes

management mode, during the same test period, the screw fastening accuracy of all three machines exceeded 99 %, with perfect 100 % accuracy achieved in multiple time periods. Throughout the entire test, the Kubernetes management mode demonstrated higher stability and achieved 100 % accuracy in several intervals, clearly showing the significant advantage of the Kubernetes management mode in improving fastening accuracy.

Table 5. Accuracy of screw locking in Kubernetes management mode

	Screw machine 1			Screw machine 2			Screw machine 3		
Time (H)	Number of screws paid (pieces)	Number of screws missing (pieces)	Number of screws paid (pieces)	Number of screws paid (pieces)	Number of screws missing (pieces)	Number of screws paid (pieces)	Number of screws paid (pieces)	Number of screws missing (pieces)	Number of screws paid (pieces)
0-0.5	188	1	99.47	204	1	99.51	208	2	99.04
0.5-1.0	180	0	100.00	224	0	100.00	196	0	100.00
1.0-1.5	176	2	98.86	220	1	99.55	204	0	100.00
1.5-2.0	196	1	99.49	204	1	99.51	200	0	100.00
2.0-2.5	188	0	100.00	220	0	100.00	192	1	99.48
2.5-3.0	176	0	100.00	212	0	100.00	212	0	100.00
3.0-3.5	208	2	99.04	200	1	99.50	192	1	99.48
3.5-4.0	212	1	99.53	164	0	100.00	224	1	99.55
4.0-4.5	204	0	100.00	220	0	100.00	176	0	100.00
4.5-5.0	196	0	100.00	204	0	100.00	200	0	100.00
5.0-5.5	184	1	99.46	208	1	99.52	208	1	99.52
5.5-6.0	188	1	99.47	208	2	99.04	204	0	100.00



a) The number and accuracy of screw lock before introducing QRM scheduling algorithm

b) The number and accuracy of screw lock after QRM scheduling algorithm is introduced

Fig. 25. Comparison of the number and accuracy of screw locks

Through comparative analysis, it can be concluded that the Kubernetes management mode

shows a significant advantage over traditional manual management in improving screw fastening accuracy. Its automation and intelligent scheduling capabilities result in more stable fastening accuracy, achieving 100 % accuracy in most test periods. This effectively reduces human errors, enhances production efficiency and product quality, and demonstrates the potential of automation management technology to improve production stability and reliability in modern manufacturing.

5. Conclusions

This paper designs a Kubernetes-based cluster management system for automatic screw machines. Machinekit is deployed in LXD containers with verified low overhead, and an improved QRM real-time scheduling algorithm is proposed to optimize task allocation and real-time performance for periodic, deadline-sensitive screw fastening tasks. Repeated experiments with controlled variables and statistical analysis show that the system increases production efficiency by 42.86 %, improves fastening accuracy to 99.62 %, reduces the number of missed screws and variance significantly, and achieves 0 % deadline miss ratio. The system provides a practical solution for intelligent cluster control of automatic screw machines and supports the digital transformation of discrete manufacturing. Future work will support multi-Master high-availability deployment, integrate HA etcd and external load balancers, and access more types of motion control equipment to expand the application scope of the system.

Acknowledgements

The authors have not disclosed any funding.

Data availability

The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Author contributions

Li-Gong: conceptualization, methodology, software, investigation, data curation, writing-original draft preparation, visualization. Yezhong-Han: project administration, resources, validation, writing-review and editing. Hehuan-Peng: supervision, formal analysis, funding acquisition, review and editing.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] Y. Liu and J. Lu, "Environmental conditions and promotion mechanisms for the optimization and upgrading of China's industrial structure under the new normal," *Zhejiang Academic Journal*, Vol. 2015, No. 6, pp. 188–193, 2015.

[2] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for Industry 4.0-based manufacturing systems," *Manufacturing Letters*, Vol. 3, pp. 18–23, Jan. 2015, https://doi.org/10.1016/j.mfglet.2014.12.001

[3] W. Dong, "Application and development of automatic screw locking machines," *China Venture Capital*, Vol. 2013, p. A18, 2013.

[4] Y. Zhang et al., "Control system of automatic tightening machine based on ARM," *Journal of Mechanical and Electrical Engineering*, Vol. 26, No. 4, pp. 47–49, 2009.

[5] L. Li and C. Shu, "Software development practice of microservice architecture and containerization technology," *Internet of Things Technologies*, Vol. 14, No. 5, pp. 64–67, 2024.

[6] Z. Wu, "Development and trend of virtualization technology in cloud computing," *Journal of Computer Applications*, Vol. 37, No. 4, pp. 915–923, 2017.

[7] M. Huang, "Design and implementation of machine learning computing platform based on container technology," *Experimental Technology and Management*, Vol. 36, No. 9, pp. 28–31, 2019.

[8] J. Zhou, P. Li, Y. Zhou, B. Wang, J. Zang, and L. Meng, "Toward new-generation intelligent manufacturing," *Engineering*, Vol. 4, No. 1, pp. 11–20, Feb. 2018, https://doi.org/10.1016/j.eng.2018.01.002

[9] Pablo Acuña, *Deploying Rails with Docker, Kubernetes and ECS*. Apress, 2016.

[10] Z. Xu, "Design and implementation of Kubernetes cluster management system for container cloud platform," Nanjing University, 2021.

[11] S. G. Dikaleh, O. Sheikh, and C. Felix, "Introduction to Kubernetes," in *27th Annual International Conference on Computer Science and Software Engineering*, 2017.

[12] S. Zuguang and X. Yuqing, "Composition analysis of miniature automatic screw locking workstation," in *Modern Manufacturing Technology and Equipment*, 2019.

[13] K. Xie and G. Yan, "Design of multi-axis controller based on Machinekit," *Journal of Mechanical and Electrical Engineering*, Vol. 33, No. 12, pp. 1471–1476, 2016.

[14] S. Böhm and G. Wirtz, "Towards orchestration of cloud-edge architectures with Kubernetes," in *Science and Technologies for Smart Cities*, pp. 207–230, 2022, https://doi.org/10.1007/978-3-031-06371-8_14

[15] H. V. Netto, L. C. Lung, M. Correia, A. F. Luiz, and L. M. Sá de Souza, "State machine replication in containers managed by Kubernetes," *Journal of Systems Architecture*, Vol. 73, pp. 53–59, 2017, https://doi.org/10.1016/j.sysarc.2016.12.007

[16] L. Osmani, T. Kauppinen, M. Komu, and S. Tarkoma, "Multi-cloud connectivity for Kubernetes in 5G networks," *IEEE Communications Magazine*, Vol. 59, No. 10, pp. 42–47, 2021, https://doi.org/10.1109/mcom.110.2100124

[17] V. Medel, R. Tolosana-Calasanz, J. Bañares, U. Arronategui, and O. F. Rana, "Characterising resource management performance in Kubernetes," *Computers and Electrical Engineering*, Vol. 68, pp. 286–297, 2018, https://doi.org/10.1016/j.compeleceng.2018.03.041

[18] G. Zhu et al., "MySQL cluster performance analysis according to pod and sharding changes in Kubernetes environment," *The Journal of Korean Institute of Next Generation Computing*, Vol. 16, No. 6, pp. 48–55, 2020.

[19] Z. Dong et al., "Research on microservice deployment based on container technology," *Information Technology and Standardization*, Vol. 2023, No. Z1, pp. 93–98, 2023.

[20] J. F. Li, C. X. Zhang, and H. Wang, "The study of a Linux container-based cloud operating system for platform as a service," *Computer Science and Engineering*, Vol. 8, No. 3, pp. 45–52, 2013.

[21] J. Kieley, "A hybrid cloud Kubernetes scheduler for machine learning workloads," Arizona State University, 2021.

[22] S. Wen et al., "K8sSim: A simulation tool for Kubernetes schedulers and its applications in scheduling algorithm optimization," *Micromachines*, Vol. 14, No. 3, p. 651, Mar. 2023, https://doi.org/10.3390/mi14030651

[23] N. D. Nguyen and T. Kim, "Balanced leader distribution algorithm in Kubernetes clusters," *Sensors*, Vol. 21, No. 3, p. 869, Jan. 2021, https://doi.org/10.3390/s21030869

[24] A. Poniszewska-Marańda and E. Czechowska, "Kubernetes cluster for automating software production environment," *Sensors*, Vol. 21, No. 5, p. 1910, Mar. 2021, https://doi.org/10.3390/s21051910

[25] D. Borsatti et al., "KubeTwin: A digital twin framework for Kubernetes deployments at scale," *IEEE Transactions on Network and Service Management*, Vol. 21, No. 4, pp. 3889–3903, 2024, https://doi.org/10.1109/tnsm.2024.3405175

[26] A. Alqarni, *Enhancing Cloud Security and Privacy with Zero-Knowledge Encryption and Vulnerability Assessment in Kubernetes Deployments*. United States: Middle Tennessee State University, 2023.

[27] C.-T. Chang and Y.-B. Lin, *Iottalk Implemented Based on Kubernetes Architecture*. Singapore: Springer, 2023.

[28] G. Coley, "Open-source hardware platform BeagleBone for electronics enthusiasts," *Electronics Production*, Vol. 2011, No. 12, pp. 5–6, 2011.

[29] L. Toka, "Ultra-reliable and low-latency computing in the edge with Kubernetes," *Journal of Grid Computing*, Vol. 19, No. 3, p. 23, 2021, https://doi.org/10.1007/s10723-021-09573-z

[30] J. Park, J. Son, and D. Kim, "Resource metric refining module for AIOps learning data in Kubernetes microservice," *KSII Transactions on Internet and Information Systems*, Vol. 17, No. 6, pp. 1545–59, 2023, https://doi.org/10.3837/tiis.2023.06.003

Li-Gong, Master's degree candidate, graduated from Zhejiang Agriculture and Forestry University's Mechanical Design and Manufacturing – Machine Automation major in June 2022, graduated from Zhejiang Agriculture and Forestry University's Mechanical major in June 2025, and joined Zhejiang Agriculture and Forestry University Jiuyang College as a full-time mechanical teacher in June 2025. Mainly engaged in research on intelligent detection and control technology. Has published 3 utility model patents, 1 invention patent, and 2 papers; won 1 first prize and 2 second prizes in national competitions; won 1 second prize, 2 third prizes, and 1 provincial-level award in provincial competitions; won 4 second prizes, 3 third prizes, and received 2 school-level awards in school competitions; participated in the compilation of the comprehensive curriculum reform textbook for industrial robot technology major in secondary vocational and higher education institutions of Zhejiang Provincial Education Department, including "Installation and Debugging of Industrial Robots System", "Mechanical Basic Knowledge and Assembly Skills of Industrial Robots" and "Mechanical Basic Knowledge and Assembly Skills".

Yezhong-Han holds a degree in Mechanical and Electrical Engineering and is a Master of Engineering and a senior lecturer. Currently, he is a member of the expert group of the Network Teacher Training Institute of Technical Schools of the Ministry of Human Resources and Social Security, an expert for evaluating the teaching quality of technical schools in Zhejiang Province, and the director of the School Development Planning Department and the head of the Postdoctoral Workstation. Has published 4 papers such as "Design and Characteristic Analysis of 2D Valve-Controlled High-Frequency Electro-Hydraulic Vibration Bench", and has obtained 3 national invention patents such as "A Leak-Proof Breathing Valve for a Pipeline Carriage", and has completed 7 national and provincial-level projects such as "Practice and Research on the Training Mode of Combinatorial High-Technical Talent in Vocational Schools". He has also published 5 textbooks such as "SOLIDWORKS Surface Application Examples Explanation".

Hehuan-Peng, Ph.D., Professor, Visiting Scholar at the University of Calgary, Canada. His main research interests are in engineering inspection and automation technology. In recent years, he has led a basic public welfare research project funded by Zhejiang Province and participated in five research projects, including those funded by the National Natural Science Foundation and the Zhejiang Provincial Natural Science Foundation. He has received one project grant for educational reform from Zhejiang Province and won a second prize for teaching achievement from Zhejiang Province Higher Education. He holds one invention patent and five utility model patents. Dr. Peng has published over 30 academic papers, with 10 indexed by SCI and 6 indexed by EI.Industry Collaboration Experience: 1. Hangzhou Tushi Information Technology Co., Ltd.: Collaborative development of "Thermal Imaging Intelligent Surveillance and Tracking System" and "3D Laser Scanning and Tracking System". 2. Hangzhou Lin'an Hangcheng Valve Co., Ltd.: Joint development of "Precision Deep Processing Technology for Metal Castings". 3. Hangzhou Longli Intelligent Technology Co., Ltd.: Collaboration in the development of an "Industrial Robot Assembly and Disassembly Platform".