Shipborne Robust Message Queuing Service Prototype System

Yongguo Jiang, Qiang Liu, Qianqian Liu, Jian Su, Changshuai Qin

Department of Computer and Technology, Ocean University of China, Qingdao, P.R.China

***Abstract*—In view of the characteristics of multi-source data fusion requirements for shipborne information systems, this thesis improves ActiveMQ and builds a shipboard robust communication prototype system. The prototype system takes advantages of reliable data asynchronous transmission to provide flexible and reliable system communication for functional platforms of shipboard information system (SIS) and overcomes the difficulties of heterogeneous system integration in SIS. This system also improves the traditional message queue model by offering multiple message queues to serve a single data link, making the native Java Message Service (JMS) more reliable. On the other hand, the system provides a scalable clustering solution that utilizes a separate configuration center to implement the expansion, control, and monitoring of message server nodes, and dynamically distribute the load of the message server and message queue. The improved system effectively overcomes difficulties of single line and single point information source failure and becomes more interference and damage resistant.**

Keywords*—*Message-oriented middleware (MOM), ActiveMQ, Java Message Service, Ship information system (SIS), System integration

I. INTRODUCTION

With the continuous advancement of marine detection technology, a large amount of information to be obtained in the fields of ship navigation, fishery fishing [1], the scientific investigation [2], and maritime law enforcement [3] is provided through advanced detection equipments At the same time, ships need to integrate a variety of detections and monitoring equipment, such as Sensor Network (SN), acoustic equipment, infrared imaging equipment, shipborne radar, Unmanned Surface Vehicles (USV), Unmanned Underwater Vehicles (UUV), Unmanned Aerial Vehicle (UAV), etc.. However, most devices are manufactured and developed by independent vendors that using different operating systems as the operating environment and relying on different network architectures [8, 9] to make the SIS a distributed application system. In the traditional centralized SIS, the direct communication between the various application systems, the connection relationship are complex. So, the traditional centralized SIS communication efficiency is relatively low [10]. On the other hand, this communication mode cannot satisfy the scalability and flexibility of the system. According to the reliability requirements of the American Bureau of Shipping (ABS) for SIS, SIS design must use redundant design to achieve reliability [9, 11]. Therefore, a reliable, flexible, and scalable heterogeneous system communication solution is needed to meet the needs of SIS integration.

Currently, enterprise distributed application system integration solutions often use message middleware (MOM) to achieve integration of heterogeneous systems. MOM provides a cross-platform asynchronous messaging solution that shields the underlying complex operating system and network architecture to ensure cross-platform data exchange between applications in a distributed network environment [12]. Most of the existing MOM products can provide general communication services, but lack of Quality of Service (QoS) support in a harsh communication environment, and can't meet the flexibility and reliability requirements of the SIS integration for the communication module. For example, MQTT [13] is a publish/subscribe protocol was designed to enable a stable communication for low power devices. MQTT is good at providing real-time and reliable data transmission in the service scenario of a bad network scenario, but often causes memory congestion when transferring large files. ZeroMQ [14], a multi-threaded network library based on the message queue, can provide high-throughput and low-latency communication services due to its agentless communication mode. However, it is not suitable for SIS applications with high reliability requirements.

In order to meet the requirements of the SIS for real-time, reliability and scalability, this paper designs and develops a shipboard embedded robust message queue service prototype system based on Apache ActiveMQ (AMQ) [15]. AMQ is an open source implementation of the Java Message Service (JMS) standard [12]. JMS includes two communication models, peer-to-peer and publish/subscribe, and provides a reliable information transfer mechanism. On the other hand, AMQ uses the message broker to manage the message queue and provides a redundant design of full backup, which realizes the clustering of brokers and ensures the reliability of information sources. However, given the limited performance of hardware devices in offshore environments, such data redundancy scale may be unacceptable. Therefore, we use a unified configuration management center to manage redundant message queues and brokers and realize the reliability and scalability of the message transmission service under the premise of ensuring redundant backup of messages.

Our main work includes the following two aspects:

A. The JMS message queue model has been improved to integrate multiple queue entities into one queue service group, each of which serves a group of producer-to-consumer data transfers.

B. The system provides a scalable clustering method, which uses the configuration center to manage the message queue entity and the message server in a unified manner, realizing the dynamic adjustment of the message queue entity in the message server and the queue service group, and guarantees the reliability of message.

II. System design

A. System integration architecture

Message-oriented middleware is an intersystem communication technique that uses the message queue mode. In a distributed system, the application (producer) that generated the message sends the specified message to the message queue. On the other hand, the application (consumer) concerned with this message accepts the message from the message queue. The delivery of the message is handled by the message queue. The producer and the consumer do not know each other's existence, and can effectively implement loosely coupled system integration.

Based on AMQ and combined with the requirements of SIS integration and application, this paper provides a fully integrated and highly survivable universal network for all mission-critical functions of SIS in the form of message queue group, including the information of control, navigation, monitoring, and detection.

B. Message Queue Prototype System Framework

The overall framework of the shipboard embedded robust message queue services prototype system is shown in Figure 1, including the configuration center, AMQ cluster, and user API. The configuration center is the core component of the prototype system. It is primarily responsible for managing and monitoring message queues, managing AMQ clusters, monitoring AMQ node status, providing the information about the message queue connection for producers and consumers, and dynamically adjusting a load of AMQ nodes and message queues. An AMQ cluster consists of multiple physical servers, each with a different AMQ broker node. Each AMQ broker node provides message queuing services, including the dump and distribution of messages. And AMQ clusters uses the backup strategy to provide SIS with highly available messaging services. The user API provides a simple programming interface for applications that can be rapidly integrated into the SIS. The process of using the prototype system for message service is roughly divided into seven steps, as shown in Figure 1.

First, the message producer asks the configuration center if the message service it needs to exist. If the service is available, the configuration center returns the message queue information used by the message service. After the message producer receives the information about message service returned by the configuration center, it connects to the AMQ cluster according to the information and sends messages to the specified node and message queue, as shown in steps 3 and 4. The consumer also needs to perform steps 1, 2, and 3 to request the information of message service from the configuration center and connect to the AMQ cluster according to the information. Then the AMQ node pushes information from the specified message queue to the consumer.

If the service requested by the producer does not exist, the configuration center registers the new service and performs step 6 to control the AMQ cluster. Meanwhile, the configuration center monitors the cluster in real time and determines whether the message service is in a healthy state, according to the node and information of message queue returned in step 7. If the message service is in an unhealthy state, step 6 is performed to dynamically adjust the cluster nodes and message queues used by the message service.

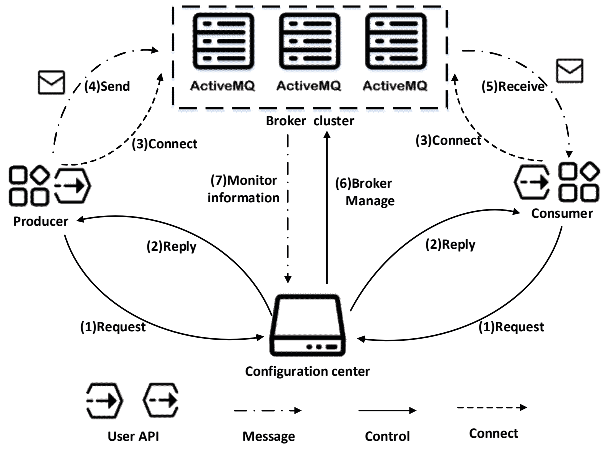


Fig. 1. Message Queue Prototype System Framework

III. Server design

A. Message queue group model

For a certain message producer, JMS uses a message queue entity to provide both peer-to-peer and publish/subscribe messaging services. A message queue entity is usually stored on a fixed message server, and producers and consumers can only send and receive messages through the message server. Even if multiple message servers are used to provide message services, the message queues of each message server are not associated. Once a message server fails, all message queues it hosts will not be available. In order to improve the reliability of the message service and meet the redundancy requirements in the SIS design, we have designed a new message queue group model as shown in Figure 2.

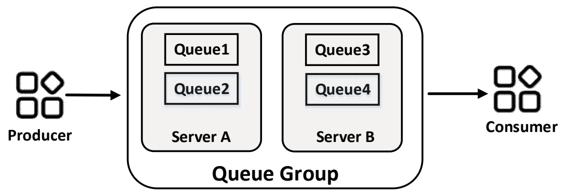


Fig. 2. Message queue group model

A message queue group is an abstraction of multiple message queue entities, containing multiple message queue entities in different message server, i.e. AMQ broker. When a message producer generates a message, it sends a message to the message queue group. Messages are sent to different message queue entities of different message servers. This means that it is not certain which message server the message is stored on. Thus, when a single message server fails, producers and consumers can perform messaging services through other message queues and servers in the message queue group. It can effectively solve the problem of single line failure.

B. Cluster strategy

Most messaging middleware products support clustering of message servers to ensure the reliability of the message source. Each messaging server in the cluster have separate message queuing services of detailed message distribution capabilities. The existing clustering scheme is divided into two categories: high-performance and high-availability.

The high-performance scheme aims to provide higher throughput in unified service time. The high-availability scheme aims to reduce the outage time of the entire message service. In the SIS integrated application environment, the primary goal of message service design is to make sure the normal operation of the service and to provide highly reliable disaster recovery performance. Existing message middleware products [15-17] provide a variety of high-availability cluster strategies, which are mainly divided into two parts: service processing node backup and data backup. Service processing node backup is used to back up the message forwarding function in the message server. Data backup is the backup of messages in the message queue. Different clustering strategies can be used in different application scenarios.

In SIS, service processing node backups and data backups are equally important. However, the backup strategy is closely related to the performance of hardware in the cluster. Considering the limited performance of hardware devices in shipboard environments, it is unacceptable to use full backups for service processing nodes and data. In order to guarantee the disaster tolerance of the service node, we hope that the service processing node uses a full backup strategy. On the other hand, message queue data use an incomplete backup strategy to reduce the redundancy of data.

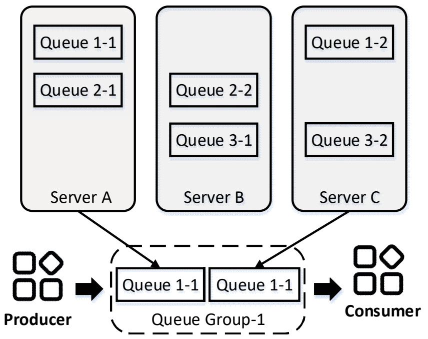


Fig. 3. Cluster strategy diagram

Combined with the message queue group model, we propose a new cluster strategy, as shown in Figure 3. Multiple message queue entities in a message queue group have the same message data, enabling redundant backup of data. This is shown in figure 3, the queue group-1 have two message queues (queue1-1 and queue1-2) that mirror each other. At the same time, in order to reduce the redundancy of data, the number of message queue entities in the message queue group (Nq) ≤the number of message servers (Ns). And these message queue entities distributed among different message servers. As an example, in figure 3, the server cluster has three message servers, i.e. Server A, Server B and Server C and three message queue groups, i.e.Queue1, Queue2 and Queue3. Each message queue group has fewer message queue entities than Ns=3, which is Nq=2. The same group of message queue entities are distributed on different servers to ensure the reliability of message transmission. In other words, this cluster strategy can still provide complete services even if one node in the cluster is damaged. On the other hand, the strategy of fully backing up message queues to all nodes in the cluster is only 50% space-efficient. The backup strategy used by the prototype system, if all message queues are double-backed, i.e. Nq=2, then the space utilization rate is at least 50% when Ns⩾3.

C. Message Server Monitoring Module

In order to monitor the real-time status of the message server and message queues, we designed the message server monitoring module. The monitoring module runs independently in each message server to obtain various information of each message server node, including system status, AMQ Broker status, and message queue status. The monitoring module is mainly divided into Java Management Extensions (JMX) management module, Sigar module, and Dubbo module which is shown in Figure 4.

The JMX management module uses the JMX extension interface provided by AMQ to obtain and control the running status information of the Broker and the message queue. Sigar is the primary data collection component of Hyperic HQ and can be used to collect information from multiple operating systems [18]. We use the Sigar tool to get system status information such as CPU, disk, network I/O, memory, and JVM. Dubbo is a Java-based Remote Procedure Call (RPC) framework that provides interface-based remote calls and automatic service registration and discovery [19].

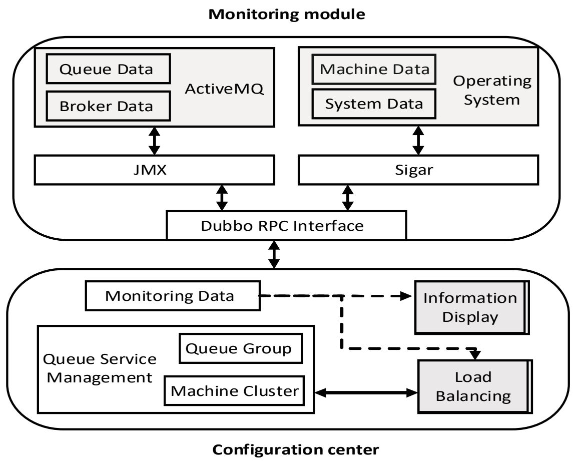


Fig. 4. Schematic diagram of the monitoring module and configuration center

The monitoring module uses Dubbo to provide remote monitoring and control interfaces for the configuration center.

The configuration center uses these interfaces to manage the message server and message queue uniformly. On the other hand, the function of automatic service registration and discovery by using Dubbo is used to achieve flexible expansion of the message server.

D. *Configuration center design*

Configuration center is the core of the prototype system, which is mainly composed of monitoring center and management center, as shown in Figure 4. The monitoring center is responsible for collecting data from each message server monitoring module and providing visualization of information. The management center is responsible for managing the distribution of message queue groups of the message server nodes and dynamically adjusting the server load. In addition, the administration center is responsible for providing message queue connection information to producers and consumers. Dynamic load adjustment refers to that the configuration center plans the number of message queues that the message server nodes undertake according to the load information obtained. The adjustment strategy depends on the monitoring data obtained by the monitoring center, such as, number of connections $B(N_i)$, number of message consumers $B(C_i)$, number of message generators $B(P_i)$, total number of messages $B(MC_i)$, proportion of memory usage $B(M_i)$, proportion of hard disk usage $B(S_i)$.

The information obtained directly from the monitoring module cannot represent the specific load situation, so a function is needed to convert the monitoring data to obtain the load status of the message server L(Si), Broker L(Bi) and queue L(Qi). Since the measurement units of monitoring data are different, and different indicators have different effects on the load of the system, the parameter r is introduced to reflect the impact of the monitoring data on the load. Take L(Bi) as an example:

$$\mathrm{L}(B_i) = [r_1, r_2, r_3, r_4, r_5, r_6] \cdot \begin{bmatrix} \mathrm{B}(C_i) \\ \mathrm{B}(N_I) \\ \mathrm{B}(P_i) \\ \mathrm{B}(MC_i) \\ \mathrm{B}(M_i) \\ \mathrm{B}(S_i) \end{bmatrix}$$

$$i = 1, 2, 3, \cdots, n, \quad (1)$$

The prototype system uses a dynamic clustering strategy to realize data backup on multiple nodes, but it also increases the difficulty of managing. The management center ensures the collaborative work of multiple service nodes through the following four important strategies:

1) *Monitoring of node survival status:*

Since calling the monitoring module requires additional system and network overhead, it is not possible to query the load information of the message server node in real time, but only once every T time interval. On the other hand, the monitoring information of T interval is used to monitor the node survival state.

2) *Election of master node:*

In a service node cluster, a master service node needs to be elected to provide external read/write services. In each message queue group, there is only one master queue. The master queue is located in the master service node. The prototype system designs the load table of the node server. The configuration center sorts the message server according to the load of the load table. When a new message queue group is created, the configuration center first allocates the message queue group to the server with less load pressure, and the master queue is placed in the server of the best choice.

3) *Copy consistency:*

A master-slave mode is used between multiple copies of the message queue. Message queue data are successfully saved by the master node and sent to other slave nodes. The configuration center acts as a coordinator to ensure the state of multiple replicas. Messages in each message queue have a unique identifier (MessageID). For each message, the unified MessageID is identified when the master server node sends the message to other replicas. When the standby server node receives a copy of the message, it returns confirmation to the node cluster and configuration center. After the message is consumed, the configuration center will listen to the corresponding MessageID information. At the same time, the configuration center removes the corresponding message copy of the slave node.

4) *Fault recovery:*

When the node fails, it is mainly divided into two types: master node failure and slave node failure.

a) Master node:

At this point, the new master node is selected. Because of the synchronous copy replication between the master node and the slave node, the slave node can quickly become the master node without having to pull unsynchronized data from other nodes.

b) Slave node:

If the slave node is failed, the synchronous replication of the message data coming from the master node will be blocked. When the slave node survival information is confirmed by the configuration center, the message of slave node failure is broadcast from the configuration center to the master node. The configuration center moves the corrupted nodes out of the cluster and redistributes the message queues within the cluster.

IV. Implementation

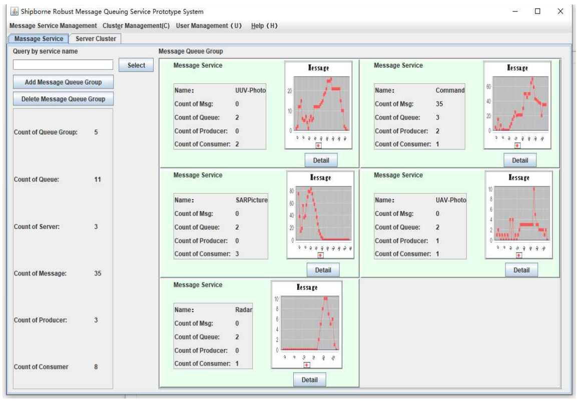
A. *Development Environment*

The prototype system was based on an x86 industrial personal computer (IPC) and was installed with Linux hairdo version CentOS 7. The message server node uses version 5.15.1 of Apache ActiveMQ, which is dependent on the Java 8 runtime environment. At the same time, in order to ensure the reliability of message data, the system uses the relational database based on MySql 5.7 to realize the local storage of message data. The RPC invocation framework used by the configuration center and monitoring module is Dubbo. The entire business layer of the prototype system was developed in Java, including the configuration center, the monitoring module, and the integrated display module.

B. *Interface design*

Display interface uses Java native Swing components for development. Machines with a Java environment can use desktop applications developed in Swing. The display module needs to display the whole function of the prototype system intuitively and provide user-friendly operation mode. According to the overall design of the system, the display interface function module is divided into message service management and cluster management, as shown in Figure 5. Due to operational security requirements, the prototype system has different permissions for different users. Administrators can manage users in the integrated interface.

Message service management provides the monitoring, registration, modification and other functions of message queue groups. The main interface provides an overview of basic information about each message queue group, including name, number of messages, number of producers, number of consumers, number of message queues, etc.. The message service detail button provides the detailed information of the message service corresponding to the message queue entity, as well as the operation of sending a message, deleting a message, suspending dispatch and so on to the message queue.



Count of Queue Group: 5
Count of Queue: 11
Count of Server: 3
Count of Message: 35
Count of Producer: 3
Count of Consumer 8

Name：	Count of Msg:	Count of Queue:	Count of Producer:	Count of Consumer:
UUV-Photo	0	2	0	2
Command	35	3	2	1
SARPicture	0	2	0	3
UAV-Photo	0	2	1	1
Radar	0	2	0	1

Fig. 5. Integrated display interface

The cluster management module takes the message server as the operating and monitoring object. This module provides the information of monitoring and basic control functions of the message server. Monitoring information includes the monitoring information of AMQ broker and server hardware. On the other hand, the management module also provides remote control functions of AMQ broker, including adding message queues, deleting message queues and so on.

V. EVALUATION

An experiment was conducted on simulation environment to evaluate the performance of prototype system and original AMQ. Testing cluster server use virtual machines environment: CentOS 7 64bit OS, single CPU core (3.30GHz) and 8GB RAM.

A. *Fault recovery*

At the same time of ensuring the reliability of message, reducing the time of system failure recovery is the key point of the shipborne robust message queuing service prototype system. To ensure that MOM continue to provide services in the event of node failure, native AMQ provides two common solutions.

1) *Network Bridge*

Original AMQ allows multiple Broker nodes to connect and transmit information to each other. This scheme can guarantee other nodes to continue to provide services if one node fails. But this scheme will not fully synchronize the data of each node, that is, the reliability of messages cannot be guaranteed.

2) *Master/slave cluster based on Zookeeper*

To enable multiple Broker nodes to provide consistent data services, AMQ provides a master/slave cluster solution based on Zookeeper [15]. ZooKeeper is a distributed coordination service provider commonly used for storage recovery [20] and load balancing [21] in distributed environments. In this scheme, ZooKeeper is used to manage broker node information and select the master node to provide external services. At the same time, slave nodes synchronize data from master nodes, ensuring that each node in the cluster can seamlessly take over.

We tested the fault recovery time for both scenarios and prototype systems without the broker keeping unconsumed massage. The test user three broker nodes and the results are shown in Table 1.

TABLE I. TIME FOR FAULT RECOVERY

Scenario	Average Time （ms）
Network Bridge	62
Based on Zookeeper Master/Salve	1854
Prototype System	687

It can be seen from Table 1 that the network bridge scheme has the fastest recovery time, while the based on Zookeeper scheme has the slowest recovery time. This time gap is due to the fact that Network Bridge only transfers the connection state, while Zookeeper requires master-slave node state comparison and primary node election. On the other hand, the prototype system does not need to compare all node states during the connection restoration process, but only needs to send the broker information where the backup message queue is located to the client. Therefore, it saves nearly 2x time compared with the based on ZooKeeper scheme

B. *Latency*

In the JMS protocol, messages are required to be stored in disk with persistence policy to ensure the reliability of messages. The prototype system backs up persistent messages stored locally to a different node to ensure that there are available copies of the messages.

However, not all consumers adopt persistence strategies, such as those service that are time-sensitive but have low data reliability requirements. The prototype system uses original AMQ to provide non-persistent message transmission service and improve the speed of message transmission. Therefore, the prototype system's non-persistent transport mode is the same as the original AMQ's non-persistent transport mode.

In the experiment, the time delay of persistent message in original AMQ system and prototype system is tested. The experiment sent data from the client every 10ms, sending 10000 message data each time, and calculated the average of each data transmission delay, as shown in Figure 6.

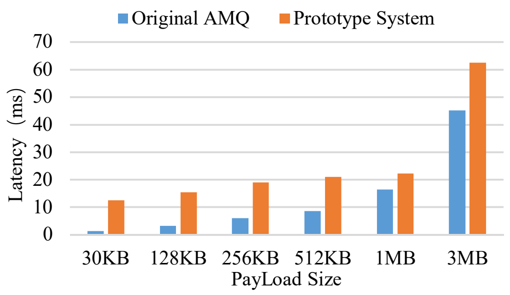


Fig. 6. Latency Results when using Non-Persistent

It can be seen from Figure 6 that the time delay of the prototype system is improved by about 10ms on average under various data load sizes. After analysis, this part of the delay appears on the Master Broker side. To ensure consistency between replicas of the message queue, persistent message queues copy the message data to the replica of the message queue at first. Then, message queue allow the message to be consumed by the consumer. Therefore, the prototype system sacrifices performance in time delay. However, with the increase of data load, the delay increase was not obvious, only about 35% with 1MB and 3MB data load.

C. *Resource Usage*

The experiment for resource usage shows CPU and disk usage.

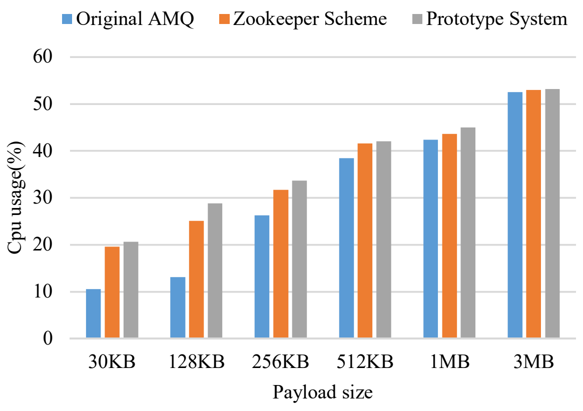


Fig. 7. Comparison of CPU usage

Figure 7 shows CPU usage as the payload size increases. The original AMQ still performs best. Comparing the two schemes of providing high availability cluster, the performance of the prototype system is not inferior to that of the ZooKeeper based scheme.

In order to compare the performance of these two kinds of high availability clusters, the experiment designed the test of disk usage. Both the ZooKeeper based scheme and the prototype system use three server nodes, i.e. Server A, Server B and Server C. In the test, three sets of message queues were used, with the producer sending 200 messages with a 1MB payload and no consumer consumption.

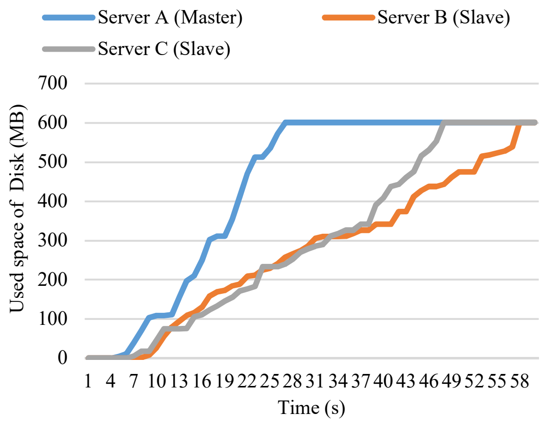


Fig. 8. Disk usage for the Zookeeper based scheme

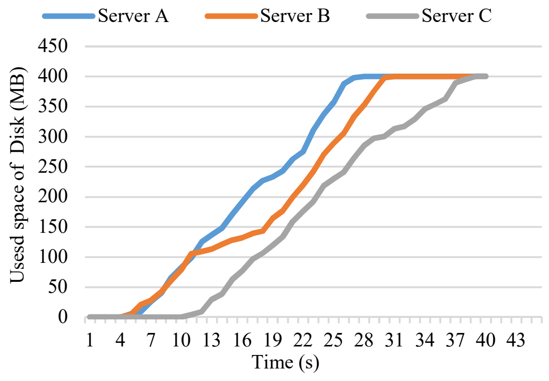


Fig. 9. Disk usage for the prototype system

Figure 8 and Figure 9 show the growth of the two schemes on the hard disk. Each node of the ZooKeeper based scheme holds data of three message queues with a total of 600MB. In this scenario, the master node receives the message from the producer and pushes it to the replica node. As can be seen from Figure 8, the master node received all the data around 25s, but the slave node needed nearly 2x time to retain all the data.

Each node in the prototype system cluster retains only two message queues with a total of 400MB. Across the cluster, there are replicas of each message queue, meaning a total of 3*2=6 message queues. In terms of disk space utilization, the prototype system is superior.

VI. CONCLUSION

The shipborne robust message queue prototype system provides a reliable and flexible solution for SIS to integrate multi-source data. The prototype system uses the message queue group model and scalable clustering scheme to reduce the hardware requirements of the message service system and provides a reference for the distributed system integration under harsh environment. However, the prototype system still needs further improvement. For example, the monitoring center and real-time warning are combined to provide users with alarm information. In addition, in the real work environment, it is not very reliable to use a single configuration center node to provide management for the entire message service, thus a configuration center clustering scheme needs to be provided.

ACKNOWLEDGMENT

This work is supported by the National Key R&D Program of China (Grant No. 2017YFC1405200)

REFERENCES

[1] M. Doray et al., "The PELGAS survey: ship-based integrated monitoring of the Bay of Biscay pelagic ecosystem," vol. 166, pp. 15-29, 2018.

[2] Y. Xie, X.-h. Yang, F.-f. Xun, and L.-y. Wang, "Integrated Data Acquisition Terminal Used on Board," in 2018 18th International Symposium on Communications and Information Technologies (ISCIT), 2018, pp. 127-130: IEEE.

[3] Z. L. Szpak and J. R. J. E. s. w. a. Tapamo, "Maritime surveillance: Tracking ships inside a dynamic background using a fast level-set," vol. 38, no. 6, pp. 6669-6680, 2011.

[4] P. S. Vincent, C. P. Gardiner, A. R. Wilson, D. Ellery, and T. Armstrong, "Installation of a sensor network on an RAN Armidale Class Patrol Boat," in Materials Forum, 2008, vol. 33, pp. 307-316.

[5] C. Jun-Hong, K. Jiejun, M. Gerla, and Z. J. N. Shengli, IEEE, "The challenges of building mobile underwater wireless networks for aquatic applications," vol. 20, no. 3, pp. 12-18, 2006.

[6] H. Ferreira et al., "Autonomous bathymetry for risk assessment with ROAZ robotic surface vehicle," in Oceans 2009-Europe, 2009, pp. 1-6: Ieee.

[7] J. Sánchez-García, J. M. García-Campos, M. Arzamendia, D. G. Reina, S. L. Toral, and D. Gregor, "A survey on unmanned aerial and aquatic vehicle multi-hop networks: Wireless communications, evaluation tools and applications," Computer Communications, vol. 119, pp. 43-65, 2018.

[8] P. A. Bernstein, "Middleware: a model for distributed system services," Communications of the ACM, vol. 39, no. 2, pp. 86-98, 1996.

[9] M. Roa, "ABS naval vessel rules (NVR) for mission critical networks, software development, and safety critical control systems," in 2007 IEEE Electric Ship Technologies Symposium, 2007, pp. 138-144: IEEE.

[10] L. Chuang, Z. Gang, and G. Q. J. C. Engineering, "Research on Data Distribution Service for Ship Information System," vol. 39, no. 9, pp. 94-97, 2013.

[11] S. Liu, B. Xing, B. Li, and M. Gu, "Ship information system: overview and research trends," International Journal of Naval Architecture and Ocean Engineering, vol. 6, no. 3, pp. 670-684, 2014.

[12] A. Benchi, P. Launay, and F. Guidec, "JMS for opportunistic networks," (in English), Ad Hoc Networks, Article vol. 25, pp. 359-369, Feb 2015

[13] A. Banks and R. J. O. s. Gupta, "MQTT Version 3.1. 1," vol. 29, p. 89, 2014.

[14] Z. Meng, Z. Wu, C. Muvianto, and J. J. I. I. o. T. J. Gray, "A data-oriented M2M messaging mechanism for industrial IoT applications," vol. 4, no. 1, pp. 236-246, 2016.

[15] B. Snyder, D. Bosnanac, and R. Davies, ActiveMQ in action. Manning Greenwich Conn., 2011.

[16] A. Videla and J. Williams, "RabbitMQ in Action," 2012.

[17] A. Toshniwal et al., "Storm@ twitter," in Proceedings of the 2014 ACM SIGMOD international conference on Management of data, 2014, pp. 147-156: ACM.

[18] VMware. vRealize Hyperic. Available: https://www.vmware.com/products/vrealize-hyperic.html

[19] Alibaba. Dubbo Development guide. Available: http://dubbo.io/Developer+Guide-zh.htm.

[20] R. Alagappan et al., "Protocol-Aware Recovery for Consensus-Based Distributed Storage," vol. 14, no. 3, p. 21, 2018.

[21] P. Hao, L. Hu, J. Jiang, X. J. C. Che, and Informatics, "A Stochastic Adjustment Strategy for Coordination Process in Distributed Networks," vol. 37, no. 5, pp. 1184-1208, 2018.