Microservice Monitoring and Handoff Algorithm Based on Dual-node Hot Standby

Xiaojian Zheng1*, Chenzhi MA2

1School of electrical and Information Engineering, Oxbridge College, Kunming University of Science and Technology, Kunming, Yunnan, 650106, China
2University of Wisconsin-Stevens Point, Wisconsin Stevens Point, 54481, United States of America
*Lxwang@ynu.edu.cn

Abstract. The service provider nodes in distributed microservice architecture often exit, join the network or have faults, which will affect the availability of the system. One of the main causes of the problem is the "bump" of service nodes. In this paper, a monitoring and switching algorithm based on dual-node hot backup is adopted, which makes the probability of bumping lower than the traditional single machine and switching method. Moreover, with the gradual increase of system scale, the performance of dual-node hot backup algorithm is still stable. Experimental results show that this method can effectively reduce the occurrence of service consumption node access failure events and improve the availability of the system.

1. Introduction

Network service presents the characteristics of "use but not own", and has uncontrollability and uncertainty[1]. Therefore, it is difficult for service-oriented network applications to guarantee continuous stability and high reliability[1]. Su Myeon Kim has monitored more than 1000 web services in UBR for two years, and found that only 34% of them are available, and 16% of them fail every week. Mike Clark also gave a similar conclusion: in UDDI, 48% of web service registration information is invalid[2].

In recent years, the rapid development of distributed microservice applications is also facing the same problem[3][4]. Due to hardware, software and network failures, or system managers let service providers (hereinafter referred to as service nodes) join or exit the network, system availability is affected. With the continuous expansion of the scale of micro service system, the problem is becoming more and more serious. How to deal with the uncontrollability and uncertainty of service and improve the availability of micro service application system has become an urgent problem. Aiming at the inherent problems of existing microservice monitoring, this paper proposes a microservice monitoring and handoff algorithm based on dual-node hot backup(DHB). By setting the parallel running primary node and backup node for each micro service, and monitoring them in the registration center, the monitoring mechanism of service consumption node is added. The algorithm improves the frequency of monitoring service nodes. Experimental results show that the average access failure of this method is reduced by 9.92% compared with the existing methods, and the system availability is improved.
2. The Bumpy Problem of Microservice

Generally, the management of microservice cluster in microservice system is completed by the registration center. The main task is to manage the joining and quitting of service nodes, and to inform the service consumption node of service changes. At present, popular registration centers, such as Zookeeper and some mainstream systems, use heartbeat detection method to obtain the online status of service nodes, that is, the registration center periodically sends detection messages to the service nodes, or the service nodes actively report heartbeat messages to the registration center, and the registration center waits for reply or heartbeat to judge the online status of the nodes. If there is no reply or heartbeat from the service node within the specified time, it indicates that it has exited from the network. The registration center will delete it from the service registry or switch its service to the backup node, and push the service change information to the service consumption node. The heartbeat of all service nodes including new service nodes must be detected.

In the traditional stand-alone backup system (SAB), the registration center detects the service node according to the set cycle, and detects the heartbeat of the service node at the end of each monitoring cycle. Since the service node exit event occurs at a certain time in the monitoring cycle, there is a detection blind area from the node exit to the end of the monitoring cycle. In this blind area, the connection information of the node in the registry is invalid, so the request of the consumption node to quit the node will fail during this period. If the node exit occurs at the beginning of the monitoring cycle, and the node switching time, the blind zone time length will even exceed one monitoring cycle. According to the usual 5-minute monitoring cycle, the monitoring blind area may exceed 5 minutes, and the probability of service request failure is very high. This phenomenon is called bumpy in this paper. Experiments show that the probability of bumps is low, but when the service node exit frequency and service consumption node access frequency increase, and the monitoring period prolongs, bumps will occur frequently.

3. Architecture and Algorithm of Dual-node Hot Backup Microservice

3.1. Dual-node hot backup microservice architecture

The micro service architecture in this paper adopts the host independent LB mode, as shown in Fig. 1, which is composed of three types of nodes, including: registration center, service node and service consumption node. The registration center discovers and manages changes in service nodes. Service nodes provide various application services. The service consumption node requests these services. Each service adopts dual duplex mode, and the primary node and backup node run the same service at the same time to ensure the overall performance and reliability, and realize load balancing and mutual backup.

![Fig. 1 Dual-node hot backup microservice architecture](image-url)

3.2. Event sequence of dual-node hot backup

The main event sequences related to monitoring service node status are as follows:
(1) The service node is added. The service node information is sent to the registration center according to the information set in the service publishing file. After receiving the message, the registration center registers the service, and then publishes the service update message to all service consumption nodes.

(2) Join the service consumption node and pull the service registry from the registration center to the local.

(3) The registration center finds that the service node exits, removes it from the registry or switches it to the backup node, and then publishes service update messages to all service consumption nodes.

(4) The registration center regularly sends monitoring messages to the primary node and backup node, and judges the status of the node according to the reply information. If the registration center finds that the service node exits, it processes according to (3).

(5) The service consumption node sends service request messages to the primary node and backup node through LB algorithm. If the service node exits, send the exit message to the registration center in time, and then process it according to (3).

3.3. Dual-node hot backup algorithm

The registry of service node configured by the registration center is mainly used to record the information of service node and backup node,

\[ R_T = \{ s_i, t_i, m_i, b_i \mid s_i \in S, t_i \in [0,1,2,3], m_i \in M, b_i \in B, i = 1,2,\ldots,n \} \]

Where S is the service set, M is the primary node set, B is the backup node set, C is the service consumption node set, ID is the available node set, and ti is the primary node and backup node status. The following are the main algorithms of the system, and other algorithms cooperate with them to complete the monitoring task.

**Algorithm 1  New service node registration algorithm**

When the registration center receives a registration message from a service node, it selects id from the ID, updates the registry RT, and sends the update registry message to the service consumption node. The steps are as follows:

1. \( m_i = \text{id}, \) select \( \text{id} \) from ID , \( b_i = \text{id}, t_i = 3 \), generate service record \( s_i, t_i, m_i, b_i \)
2. \( s_i, t_i, m_i, b_i \) is inserted into the registry RT
3. for \( c_i \) in C: # refresh the registry of service consumption node
   Send a message to \( c_i \) to update the registry of \( c_i \)

**Algorithm 2  Timing detection service node reply algorithm**

At the end of the monitoring cycle, the registration center sends a detection message to all service nodes and sets a timer, then receives the return information and caches it. When the timing interrupt occurs, the registration center checks the return status, updates the registry of the registration center and the service consumption node, and switches the node. The steps are as follows:

1. for \( s_i \) in S:
2. \( t_i = 0 \) # Detect the reply of service node
3. if \( \text{ret}(m_i) \&\& \text{ret}(b_i) \): # the primary node and backup node are online
4. \( t_i = 3 \)
5. else:
6. if \( \text{not ret}(m_i) \):
7. Select \( \text{id} \) from ID , \( m_i \) switches to \( \text{id} \), \( m_i = \text{id}, t_i = 1 \)
8. if \( \text{not ret}(b_i) \):
9. Select \( \text{id} \) from ID , \( b_i \) switches to \( \text{id} \), \( b_i = \text{id}, t_i = 2 \)
10. for \( c_i \) in C:
11. Send a message to \( c_i \) and update its registry.

---

### Algorithm 3 Detection service request reply algorithm

When a service consumption node requests a service, it first obtains the service node connection information from the registry, sends the service request message to \( m_i \) and \( b_i \), and sets the timer, then receives the return information and caches it. When the timing interrupt occurs, the service consumption node checks the return information, and then sends the service node change message to the registration center. The steps are as follows:

1. If \( \text{ret}(m_i) \&\& \text{ret}(b_i) \):  # The primary and backup nodes are online
   2. Send \( m_i, b_i \) online message to the registration center, flag = 1
   3. Else:
      4. If not \( \text{ret}(m_i) \):
         5. Send \( m_i \) exit message to the registration center
         6. Else:
             7. flag = 1
             8. If not \( \text{ret}(b_i) \):
                9. Send \( b_i \) exit message to the registration center
                10. Else:
                    11. flag = 1
                    12. If flag == 1:
                        13. Process the results returned by the service

---

### 4. Experiment and Result Analysis

#### 4.1. Experimental explanation

This section verifies the availability of the system when the service node continuously exits the network under different system sizes and the service consumption node visits the service node. One of the important design principles of microservice architecture is the low coupling between services\(^4\)\(^5\), so the experiment assumes that the services are independent of each other. Let \( A_m \) be the event "master node normal", \( A_b \) be the event "backup node normal", and \( A \) be the event "access service successful". Since the primary node and backup node of each service run in parallel, if and only if at least one node is running normally, the access to the service can be successful. Therefore, \( A = A_m \cup A_b \), the probability of normal operation of the service can be obtained by the independence of the event

\[
P(A) = P(A_m) + P(A_b) - P(A_m A_b) = P(A_m) + P(A_b) - P(A_m) P(A_b)
\]

Service availability is the possibility that services provided by service nodes can be accessed, which describes the probability of normal operation of services\(^2\). The larger the value is, the service is always available. The smaller the value is, the service availability at a certain time cannot be predicted. The measurement method based on the number of times the service works normally is adopted\(^2\), which is calculated by equation (2)

\[
A(T) = \frac{N_s(T)}{N_a(T)} \times 100\% \tag{2}
\]

Where \( N_s(T) \) is the number of successful service calls in time period \( T \), and \( N_a(T) \) is the total number of service calls in time period \( T \). For example, assuming that the probability of normal operation of nodes is equal, the service availability is calculated by formula (1), and table 1 is obtained. The availability of a system composed of multiple microservices is described by the mean value of each service.
Table 1 An example of service dual-node hot backup availability

| Probability of node working normally | Service availability |
|--------------------------------------|----------------------|
| 0.34                                 | 0.564                |
| 0.48                                 | 0.73                 |
| 0.6                                  | 0.84                 |
| 0.8                                  | 0.96                 |
| 0.9                                  | 0.99                 |
| 0.98                                 | 0.9996               |

4.2. Experimental verification

This paper compares DHB algorithm with SAB algorithm to test their usability. In the simulation environment, the processor is Intel(R) Core(TM)2 Duo CPU P8600 @2.40GHz 2.40GHz, the memory is 2.0GB, the operating system is Windows Vista Home Basic, and the test platform is programmed with Python 3.9.0. The test parameters are set as follows: the frequency of node joining and exiting is 1 batch / S; the frequency of service request is 10 times / S; the monitoring cycle is 1 second; the scale of service node is 10, 20, 30, 40, 50, 60, 70, 80, 90, 100; the test time is 100 seconds. The test results are shown in Fig. 2.

![Fig. 2 Comparison of DHB and SAB algorithms](image)

a. Node normal probability: 0.34  
b. Node normal probability: 0.48  
c. Node normal probability: 0.9  
d. Node normal probability: 0.98

The above test results show that both algorithms can produce bumpy, but the frequency and change of bumpy are different. In the same scale, the availability of DHB algorithm is 9.92% higher than that of SAB algorithm, and the highest is 26.36%. With the increase of the number of service nodes, the availability value of DHB algorithm decreases slightly, the consistency is good, and the change is stable, while the availability value of SAB algorithm decreases significantly. With the continuous expansion of the scale of micro service system, the performance degradation is more serious.

5. Conclusion

In this paper, we find that one of the main factors affecting the availability of micro service system comes from the bumping of service nodes. The root of the problem is that the heartbeat detection
method has a blind spot when dealing with the exit and sending failure of service nodes. The experimental results show that the probability of bumps of the proposed algorithm is less than that of the single machine algorithm, and the performance of the dual-node hot standby algorithm is more stable with the increase of the network size. Of course, because this method needs to set the primary node and backup node for each service, the system overhead is doubled at the same time.

Acknowledgments
This work was financially supported by Yunnan Provincial Education Department Scientific Research Foundation of China under Grant No.2021J0839 and 2010C011.

References
[1] Wen Yan, Fang Jun, Liu Chen. An Approach to Improve the Service Availability on the Basis of Business Service Abstraction[J]. CHINESE JOURNAL OF COMPUTERS, 2010, 33(11): 2190-2201.
[2] Feng Xiu zhen, Wu Gaofeng, Jin Yueping, et al. Availability-guaranteed model for service discovery[J]. Computer Engineering and Design, 2011, 32(8):2692-2694.
[3] Knoche H, Hasselbring W. Using microservices for legacy software modernization [J]. IEEE Software, 2018, 35(3): 44-49.
[4] Dai Fei, Liu Guozhi, Li Zhang, et al. Microservices: Architecture, Communication, and Challenges[J]. JOURNAL OF APPLIED SCIENCES, 2020, 38(5): 761-778.
[5] Pahl C, Jamshidi P. Microservices: a systematic mapping study [C]// Proceedings of the 6th International Conference on Cloud Computing and Services Science, Rome, Italy, 2016: 137-146.