A Distributed Elastic Messaging System

Jiayuan Sun [a], Hua Zhang [b]

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[a]4321sunjiayuan@163.com, [b]zhanghua_288@bupt.edu.cn

Abstract. The distributed message queue can ensure the stability and reliability of the asynchronous message communication between modules, and simplify the complexity of the system. In this paper, a distributed message system architecture is proposed, which can make the message system have the ability of elastic load and realize a message system based on the structure. Experiments show that the message system can adjust the number of modules dynamically according to the load, balance the load between the modules, and realize the elastic load well.

Keywords: Messaging System; Elastic; Distributed.

1. Introduction

Message queue can improve the stability of the distributed system, reduce the coupling degree of the system and increase the scalability. Because of the limited computing resources, the message queue should have the ability to take and release resources dynamically in order to make full use of the limited resources. Therefore, how to allocate appropriate resources for message queues in the case of load fluctuation is a difficult problem, which requires the system to have the ability of elastic load.

There are many different structures for message queuing of elastic load. EQS [1] is a message queue based on ZeroMQ, which can dynamically acquire resources according to the load, but cannot release resources quickly when the load reduced. Reference [2] is also an elastic message queue structure, but a queue read and write must be in the same module.

This paper proposes a distributed message system structure, which can make the message system has better elastic load capacity. It has three kinds of modules, namely front module, storage module and management module. Front module receives user's commands and forwards these commands to storage module. Storage module inserts or returns messages according to the received commands. Front modules and storage modules send heartbeat packages to management module, and heartbeat packages contain the load of modules. Management module uses the load of modules to judge the status of the system and balance the system. Compared with other elastic message system, load balance of the message system is more flexible. Queues can change the modules they read from or inserted in dynamically according the load of the modules. A queue can store its messages in different storage modules, and can read and write at same time in two different modules.

2. System Structure

This section presents the system structure. It ensures that the elastic load of the message system, as shown in Figure 1.

2.1 The Overall Structure.

Message system consists of three kinds of modules, namely front module, storage module and management module. Each module can be deployed on machines in different geographical locations, so that the system can still operate in all kinds of unexpected situations.

2.2 Consumer / Producer.

The consumption and production module is a module that users directly use. Users use these modules to send messages to message queues or receive messages from message queues.

2.3 Front Module.

The front module receives the commands of the consumption module and the production module, and gets a message from the storage module or sends a message to the storage module according to the command. The front module can locate the storage module where a queue currently in based on the queue name in the command, and send the command to that storage module. The front module will cache the queue name and the corresponding storage module address. If front module receive a command which its queue name has not appeared in the cache, the front module will ask the management module for queue address. The front module needs to send heartbeat packets to the management module.

2.4 Storage Module.

The storage module is used to store the queue name and the messages in the queue. These storage modules perform message dequeuing and enqueueing tasks independently without cooperation or contact, which makes it easy to dynamically add or delete storage modules. The storage module also needs to send the heartbeat packet to the management module in a certain interval. The heartbeat packet includes the address and load of the storage module to help the management module's load balance operation. The messages in the storage module are organized into different lists according to the queue. The new messages are stored at the end of the list, and the first message of the list is returned when the message is read.

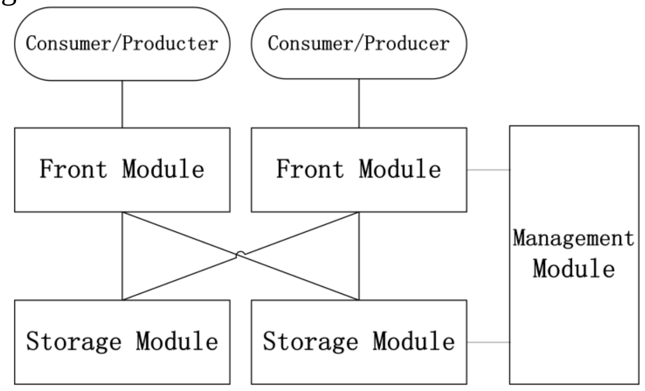

Fig. 1 The distributed structure of the message system

2.5 Management Module.

The management module is used to monitor the running status of the entire system, schedule the resources and control the address where queues are inserted in. The management module decides when to add or remove modules and controls the load balance between modules. Management module also stores positions of the queues. The positions of a queue include the insertion position and the reading position. After management module executes the load balance algorithm, the inserting address of some queues will be changed. These changes are notified to the front module by the management module. For efficiency reasons, after the queue insertion position is updated, the original message that has been inserted into the storage module is not transferred to the new insertion position. So the reading position of the queue is not changed until messages in reading position are read completely, and the queue can achieve the read and write operation separation. The management module will use the linked list to store the position of the queue. The first of the list store the position where the queue reads from and the end of the list store the position where the queue inserts in. Once an inserting position is updated, the linked list will add a new position at its end.

3. System Implementation

This section describes how message systems implement load balance and elastic load.

3.1 Elasticity Algorithm.

The load state of the system is divided into three types, idle, normal, overload. The system state is calculated by the total load of the system.

Judgment formula is as follows:

$$Avg = \frac{1}{n}\sum_{1}^{n} x_i.$$

$$\text{System State} = \begin{cases} \text{idle},\ Avg \le Lmin \\ \text{normal},\ Lmin < Avg \le Lmax. \\ \text{overload},\ Avg > Lmax \end{cases}$$

It should be ensured that

$Lmin < Lmid < Lmax.$

Where *Avg* is the average load of the system, *n* is the number of storage module, *Lmin* is the dividing line between idle and normal, *Lmax* is the dividing line between normal and overload, *Lmid* is the load of the system we expect, and x_i is the load of the *i*-th storage module. The load of the storage module can be determined by memory, CPU usage, storage space and network bandwidth. The management module calculates the number of modules required by the system, denoted as *m*, that could make the system average load is *Lmid*.

Next, calculating the number of storage modules required by the system:

$m=\mathrm{ceil}(\frac{1}{Lmid}\sum_1^n x_i).$

To make the number of storage modules is equal to *m*, if the system is in overload state then start some new storage modules, if the system is in idle state then select some lowest load modules and then set these modules as release status.

3.2 Load Balance Algorithm

After determine the system status, the next step is to determine the state of each storage module. Judge each storage module which is not in release status, module load state is divided into idle, high load and overload.

$$\text{Module State} = \begin{cases} \text{idle}, & Lu \le Lr \\ \text{high load}, & Lr < Lu \le Lh. \\ \text{overload}, & Lu > Lh \end{cases}$$

The *Lu* is stored module load, *Lr* is the dividing line between idle and high load, *Lh* is the dividing line between high load and overload. When the storage module is in the idle state, it means that the module has sufficient load free to accept the new queue. When the storage module is in a high load state, the load of this module is already high. It can maintain the current demand, but cannot add new load. When the module is in the overload status, indicating that the module has too much load, the module need to reduce part of the load. *Lr* and *Lh* must satisfy that:

$$Lmin < Lmid \le Lr < Lmax <= Lh$$

Next, allocate a new inserting position in idle modules for queues which inserting address is in a release module.

Inserting address should be allocated to all idle modules according to the load of idle modules. In the case of a large number of queues, accurate insertion position assignments based on queue traffic and module load requires a lot of computation. For efficiency reasons, a random algorithm with weights is used to quickly locate inserting position. The weights are calculated from the load of the module. For each idle storage module, find a number rangedfrom0 to1. When the queue is assigned, calculate a random number between 0 and 1. The random number in which module's range, the queue inserting position will reassign to that module.

$r_i = \frac{1}{x_i}\sum_0^n x_j,$

$Al_i = \sum_0^{i-1} r_j \,/\, \sum_0^n r_j,$

$Ar_i = \sum_0^i r_j \,/\, \sum_0^n r_j.$

In these formulas, r_i is the probability is that *i*-th idle storage module receives a queue and *n* is the total number of the storage modules in the idle state. Al_i is the lower bound of *i*-th storage module, and Ar_i is the upper bound of the *i*-th storage module.

Overload modules also need to change part of queue's inserting position to reduce load. For each queue in overload modules, determine whether to transfer:

$$p = (Lu - Lmid)/Lu.$$

Lu is the load of an overload storage module, and p is the probability of the queue needs to be transferred. Find a random number between 0 and 1, and if it is greater than p, the queue transfers the storage module in the same way as above.

In the case of a large number of queues, sorting based on load or looking for a larger load in the queue also takes a lot of computing resources. The random allocation algorithm can well balance the load between modules, and avoid the sorting or query performance.

3.3 Insert Message

The user uses the producer module to transform the message into a command and sent it to front module. The front module looks in its cache for the address of the queue where the message is inserted. If there is no queue name in cache, then ask management module for an inserting address. If the address of the queue is stored in the management module, the address is returned directly. If no address is stored, the storage module with the lowest load is selected as the address. When the management module modifies the queue insertion address, it will synchronize the new address of the queue to all front modules.

3.4 Read Message

Front module receives command sent by consumer module, and then finds the queue reading address in local cache. If there is no record, Front module asks the management module for reading address. If the management module has the address, it returns to the front module; if not, it returns an invalid queue name error to the front module. The front module requests the data from the storage module according to the address, and if there is no message in the corresponding queue in the storage module, it returns null to the front module. The front module requests the new address of the queue from the management module. If the returned address is not the same, the front module gets the message of the queue from the new address. If so, the front module returns null to the user.

4. Experiment

In this section, we test the performance of the system. These tests are run on a Linux system computer, the main configuration of the computer is 2G memory and i3 CPU. The load of the system mainly comes from sending and receiving messages. In order to facilitate the experiment, we use the number of messages the system sent and receive as the system load criterion.

Table 1 Message system performance

Message Length / **Machine Configuration**	**100B**	**512B**	**1KB**	**10KB**
1Storage Module 1Queue	13468	11098	7448	2364
2Storage Module 1Queue	25734	22016	13634	4272
2Storage Module 2Queue	26544	23018	14962	4532

Table 1 shows the performance of the system in three cases: (1)A queue access to the system when the system has a storage module on a machine. (2)A queue accesses to the system in the case where two machines are deployed two storage modules. (3)Two queues access to the system in the case where two machines are deployed two storage modules.

Comparing (1) and (2), we can conclude that the two storage modules are much faster than a storage module in the case of a queue and a front module. Because in the case of a large load a queue's read and writes operation can be completed in different modules, the performance has greatly improved. Comparing (2) and (3), it can be seen that they have nearly same performance. In case (3), there are multiple queues, both machines are heavily loaded and do not perform load balancing. This is the load limit of the system. In case (2), load balancing of queues may consume some resources, but they do not have much impact on performance, so it is close to the load limit.

We use the number of messages as the basis for load judgments, set some parameters to observe the implementation of the system load balancing process, set $Lmin$=100， $Lmid$=300， $Lmax$=500， Lh=500， Lr=400. Using four front modules and four storage modules, 20 different queues are divided into four groups. Each group is on a front module, and each group begins to insert messages in an interval of 20 seconds. The inserting rate is 0 per second at beginning, and increase 2 per second until 20 second after begins. After 160 second from begin, queues decreases 2 per second until 0 per second.

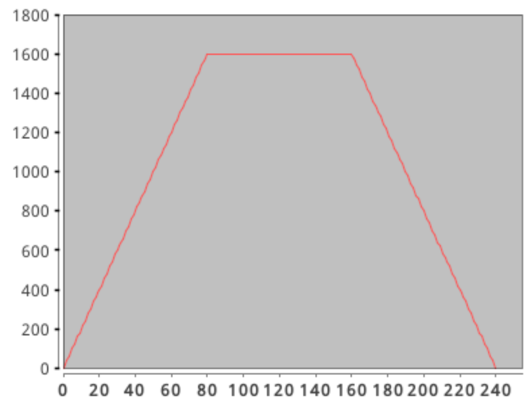

Fig. 2 Total load of the system

Figure 2 shows the system load changes over time, X-axis is the time the system runs in seconds, Y-axis is the number of messages through the system. Figure 3 show the load of the four storage modules. It can be seen that as the system load increases, the original module is not enough to meet the performance requirements, four storage modules will be started one by one. The excess load will be transferred to the new module. The load between the four modules remains stable when the load does not change. When the total load decreases to a certain degree, the system will release a module, and the load on the module is transferred to other modules.

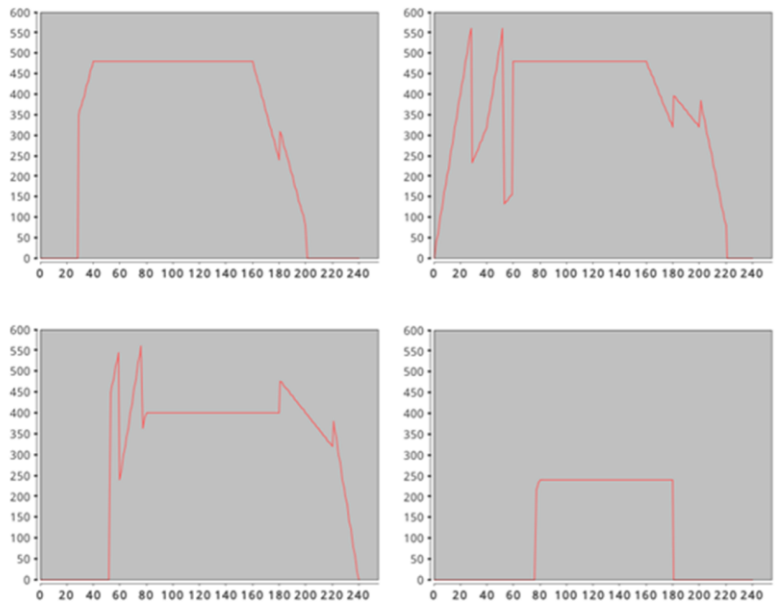

Fig. 3 The load of storage modules.

5. Summary

We proposed a distributed elastic messaging system, and shown the performance of this system, it can maintain a good elastic load capacity without impact on the performance. This system has three main advantages: (1) It will use the resources that have been got as far as possible, until the resources are not enough. (2) Using a random way to distribute the load, to avoid too much load in one module in a short time, which resulting in secondary load balancing. (3) When the system needs to release resources, it can quickly transfer the load to the other modules. The next step, we will study the message queue persistence for the elastic load.

Acknowledgments

This work is supported by NSFC (Grant Nos. 61300181, 61502044), the Fundamental Research Funds for the Central Universities (Grant No. 2015RC23).

References

[1] Tran N L, Skhiri S, Zim´Nyi E. EQS: An Elastic and Scalable Message Queue for the Cloud [C] // IEEE, International Conference on Cloud Computing Technology and Science, Cloudcom 2011, Athens, Greece, November 29 - December. 2011:391-398.

[2] El Rheddane A, De Palma N, Tchana A, et al. Elastic Message Queues [C] // IEEE, International Conference on Cloud Computing. 2014:17-23.

[3] Pietzuch P, Eyers D, Kounev S, et al. Towards a common API for publish/subscribe. [C] // Inaugural International Conference on Distributed Event-Based Systems, Debs 2007, Toronto, Ontario, Canada, June. 2007:152-157.

[4] Chen J, Soundararajan G, Amza C. Autonomic Provisioning of Backend Databases in Dynamic Content Web Servers [C] // IEEE International Conference on Autonomic Computing. 2006:231-242.

[5] Patel D, Khasib F, Sadooghi I, et al. Towards In-Order and Exactly-Once Delivery Using Hierarchical Distributed Message Queues [C] // IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. 2014:883-892.