HBase Connection Dynamic Keeping Method Based on Reactor Pattern

HBase Connection Dynamic Keeping Method Based on Reactor Pattern

Lai Xinming[1], Wang Haitao[2], Zhao Jing[2], Zhang Fan[2], Zhao Chao[2], Wu Gang[2,*]

[1]Aisino Corporation, Beijing, China

[2]China National Institute of Standardization, Beijing, China

*Corresponding author

Email: wugang@cnis.ac.cn

Abstract. This paper proposes a HBase connection dynamic keeping method based on the Reactor mode. This method uses the Reactor mode to implement the client connection maintenance mechanism based on the NSQ message queue. By only creating the TCP control block of the client connection and buffering it to the NSQ queue, it improves the response of the user connection and the concurrent processing capacity. By monitoring the active state of the connected TCP block in the message queue, the real-time processing capability of the connection is guaranteed. The reuse of HBase connection objects is realized through the listen-back callback mechanism, and the CPU and memory resource utilization is improved by dynamically adjusting the size of the connection object pool.

1. Introduction

With the improvement of big data technology, more and more enterprises have started to use the Hadoop platform to manage their T-level PB-level data resources. HBase is a columnar database based on HDFS, which improves the processing performance of some business data by dozens of times based on the characteristics of columnar storage. Based on the characteristics of HDFS distributed storage, it has high data reliability even if it stores massive data.

It is precisely because HBase is applied in the large data mass storage scenario, HBase will also be used by a large number of users at the same time, so in addition to providing higher data storage performance, it also needs to provide higher data access performance. In the HBase architecture design, the client obtains the HRegionServerA where the Meta table is located through Zookeeper, and then connects to the HRegionServerA to obtain the HRegionServerB where the required data is accessed, and finally establishes a connection directly with HRegionServerB and performs data interaction. The original intention of this architecture design is to enable all accesses to be balanced across all servers to improve the ability to cope with high concurrency scenarios. But this is very dependent on the perfect table structure design. Only the RowKey of the HBase table is designed very well, so that the common hotspot data can be scattered on various servers to achieve the goal. This is usually difficult to do, especially when the HBase table is initially created and after the data is imported in batches, the latter is an operation often encountered in big data scenarios. Therefore, a large amount of data access will eventually be concentrated on several servers. While the server respond out time, the creation of a large number of connection requests will exhaust node resources and eventually cause downtime, which will cause HMaster data migration tasks and will consume cluster performance. ,Which further affects the user's access to the HBase database, and gradually forms an avalanche effect that makes the

HBase service unavailable. HBase failures caused by a large number of concurrent requests include the following aspects:

(1) A large number of short-term concurrent requests make it difficult for the HBase server to listen to the process and cause service downtime.

(2) Continue to request connections in large increments. The HBase server continuously consumes server resources in creating and destroying connections, which affects the access performance of the HBase database.

(3) When there are a large number of users connecting at the same time, the HBase server needs to maintain a connection to each client and continuously poll for I / O operations listening for the connection. A single server is limited by physical resources, so its serviceability is limited, and it cannot support a large number of user accesses, which makes the service periodically unavailable.

In order to solve the above problems, it is necessary to effectively respond to high concurrent connection requests and prevent client connections from exhausting server resources. The usual practice is to create an HBase connection pool in advance on the server side, and allocate a connection from the connection pool when there is a new connection request, and put the connection back into the pool when the client disconnects. The advantage of this approach is to save connections the creation overhead, when there is a high concurrency demand, you can use the existing connection. However, determining the size of the connection pool is a very complicated issue. If the connection pool is too small, high-concurrent requests for connections still need to be created and external connections will still cause downtime. If the connection pool is too large and a large number of connections are not used, A huge waste of resources. And the resources of a single physical machine are limited, and the maximum connection pool size that it can support is also difficult to support a large number of user connections.

2. Methodology

2.1. Overview of Logical Architecture

Aiming at the HBase database connection problem, based on the massive user concurrency scenario of the big data background, a method for dynamically maintaining HBase connection based on the Reactor pattern was proposed. The method mainly includes a connection processing main program mainProcessor implemented based on the Reactor pattern, a connection distribution subroutine subProcessor, and a connection cache module NSQQueue implemented based on NSQ. The connection processing main program includes a callback module connCallback, a connection authentication module connAuth, and a business processing module connTrans. The connection distribution subroutine includes HBase connection pool management module poolManager and connection processing module connProd.

Run mainProcessor and subProcessor on each HBase server. The mainProcessor's connTrans is responsible for listening to the HBase client connection request from the server. After receiving the connection request from the client, call connAuth according to the configuration to authenticate the client information. After the authentication is passed, the operation The system api creates a server-to-client TCP connection block and inserts the TCP connection block into the NSQQueue to make it active. The poolManager load in the subProcessor creates a configurable HBase connection pool during the initialization phase and manages the connection pool in the subsequent. ConnProd is responsible for consuming the TCP blocks in the NSQQueue and assigning an HBase connection object through the poolManager. The client information of the block is registered in the HBase connection object through the poolManager. At this time, the client has obtained the connection object of the HBase server and can access data normally. After the access processing is completed, the poolManager is responsible for recycling the HBase connection object. It will call back the registration information in the HBase connection object and return the processing result information of the connection access. At this time, the connCallback in the mainProcessor receives the processing result message, returns the processing result information of the connection to connTrans, and finally returns the message to the client through the TCP connection. At the same time, tonTrans sets the TCP block in NSQQueue to inactive. When connTrans listens to the client disconnecting, it will call the operating system API to delete the client's TCP block from the NSQQueue. The overall system architecture is shown in Figure 1. Each module is described below.

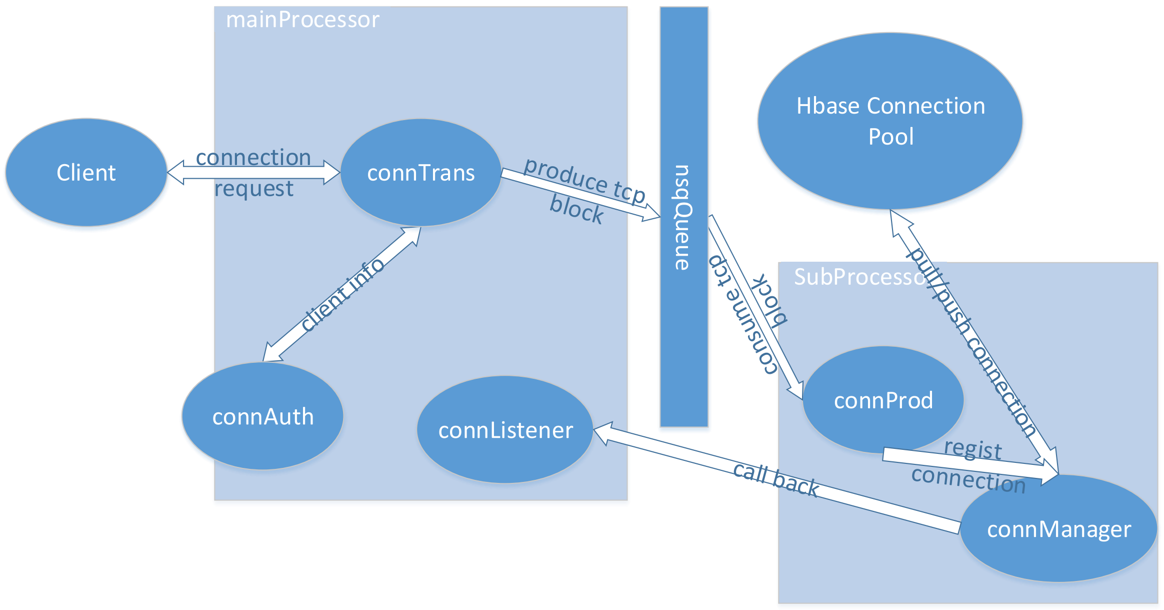

Figure 1. The diagram of the overall system architecture

2.2. *Function Modules of the System*

2.2.1. Business processing module connTrans The main listener implemented based on the Reactor pattern is responsible for monitoring and maintaining the connection requests of all clients, calling the operating system api to create a TCP control block associated with the client, and returning the final result to the client after the client's business processing request is completed.

2.2.2. Connection authentication module connAuth It mainly cooperates with the business processing module to authenticate client information. Receive the client information sent by the business processing module, and authenticate the client to the business processing module anyway.

2.2.3. callback module connCallback
When connProd consumes the TCP block in NSQQueue and allocates it to the HBase connection object through the poolManager, it will register callback information to the connection object. When the connection object is recycled, it will return some result information through the callback module. The registration information includes TCP information and client information, so the result information can be fed back to the client through connTrans.

2.2.4. Connect cache module NSQQueue The connection cache module is based on the NSQ message queue technology. It saves all the TCP control blocks created by the current operating system and connected to the client, and stores status codes and client-related information for each TCP control block. Client connection request.

2.2.5. Connection processing module connProd Responsible for sensing client connections that need to be processed and consumed in NSQQueue. For each active client connection TCP block, connProd requests to allocate an HBase connection object from the poolManager. At the same time, connProd will count the active TCP blocks in the NSQQueue to be consumed, calculate the current connection pool processing pressure, and apply to the poolManager to temporarily expand the HBase connection object pool when needed.

2.2.6. connection pool management module poolManager Responsible for maintaining and managing the HBase connection object pool, and dynamically taking connection objects from the connection object pool according to demand. And can dynamically adjust the size of the connection object pool according to demand.

3. Conclusion

The proposal of this paper quickly responds to client connection requests and improves user experience when HBase has high concurrent connections; it can greatly improve the performance of HBase and increase the size of users that a single server can support; using the method of the present invention can also make full use of HBase connection object resources Improve physical resource utilization. The key point of the solution proposed in this paper proposed are:

(1) In the high-concurrency connection request scenario, the connTrans business processing module based on the Reactor pattern does not directly create HBase connection objects for client connection requests. Instead, it creates a TCP control block for the client request only through the operating system API, and then handles the connection through connProd. The module is designed to greatly reduce resource consumption, prevent exhaustion of CPU and memory resources under high concurrent requests, and improve concurrent processing performance.

(2) In a large-scale connection request scenario, instead of allocating HBase connection objects for each connection request, a message queue is used to cache connection requests. Only active connections that require business processing are allocated connection objects from the HBase connection object pool. After the service processing ends, the TCP control block status of the connection request is set to inactive until the client connection is disconnected. Each connection object can process the business of multiple clients, which greatly improves the interest rate effect of the connection object and improves the processing performance of HBase's large-scale requests.

(3) By sensing the pressure of the number of active connections in NSQQueue, you can dynamically expand the size of the connection pool, improve server resource utilization, and prevent client connection timeouts caused by too small connection pools and resource waste caused by excessive connection pools.

Acknowledgment

This paper is supported by grants from National Key R&D Program of China (522016Z-4973) and China National Institute of Standardization (522018Y-5941, 522018Y-5948).

References

[1] https://hbase.apache.org/.

[2] Zhengjun, Pan, Lianfen, Zhao. (2018) Application and research of massive big data storage system based on HBase. In: 2018 3rd IEEE International Conference on Cloud Computing and Big Data Analysis, ICCCBDA 2018, p 219-223.

[3] Wei, G., Shengmei, L., Wenhui, Z, Di Tang Yun, Z., Juan, Z., Wen Wu Q., Chunfeng Y., Yihua H. (2016) HiBase: An efficient HBase query technology and system based on hierarchical index [J]. Computer Journal, v 39, n 1, p 140-153.

[4] Heng, Q. (2016) Research query and indexing mechanism based on HBase [D]. Chongqing University of Posts and Telecommunications.

[5] Bao, Congkai, Cao, Meiyang. (2019) Query Optimization of Massive Social Network Data Based on HBase. In: 2019 4th IEEE International Conference on Big Data Analytics, ICBDA 2019, p 94-97.