Integration of Raft and RUDP to GaussDB

Table of Contents

1 Introduction

This report set out to introduce the integration of Raft protocol and RUDP communication protocol into Gauss MPP DB that will aid in advancing the replication procedure.

Gauss MPP DB (massively parallel processing database) is a distributed database system that is scaled to a cluster of nodes, designing to speed the performance of dealing with massive amounts of data. Aiming at masking the node vulnerability, it features master/slave architecture, WAL log-shipping replication and state restoring with the anticipation that the system can recover quickly in the case of the master failure. To improve the efficiency of failover, a Raft-based replication model in Gauss MPPDB. Without the intervention from CM in the view change and the clumsy copy during the log backup, raft replication can be competitive with lower overheads to the monitoring and faster recovery from failures.

Besides, aiming at improve the throughput in case of the network latency and frequent packet loss, RUDP is applied to Raft. the Reliable User Datagram Protocol (RUDP) is a packet based transport layer protocol aiming to provide a solution where UDP is too primitive but TCP adds too much complexity. In order for RUDP to gain higher quality of service, RUDP implements features that are similar to TCP with less overhead.

2 Integrate Raft Consensus Protocol into GaussDB

Individual host or node in the distributed database system is vulnerable to data loss due to physical damage or power breakage; thus replicas are usually used as insurance against hardware and software fault. The consequent data consistency problem arises. Consensus is a key component to providing fault-tolerant services such as replicated data stores and raft is among the most popular consensus algorithms. Raft protocol allows a collection of machines to work as a coherent group that can survive the failures of some of its members.

It should be noted that the failure of data-node is detected by the CM agent instead of by the cohorts group itself. Undoubtedly, the monitoring and arbitration will bring pressure to bear on the cluster management. As the database expands, the program is becoming more serious.

Another doubt comes with the log copy from one node to its backups. Not only inefficient as the transferring is, but also a data loss may happen during this process if configured with asynchronous commit.

The following will talk about those situations in details and explain the corresponding strategies as suggested in Raft protocol.

2.1 Limitations of traditional replication in GaussDB

2.1.1 Limitations of Master-Slave architecture

In GaussDB, the abnormal status of each datanodes is monitored by CM agent and role switch is controlled by the CM server. The CM agent cooperating with a certain data-node moniters the instance's status and report it to the CM server. If an instance failure is caught, CM server will invoke corresponding actions and pass it down to the CM agent. Then the CM agent will , the standby will be taken up to act as the primary. Finally, the cm server will record the role change in the configuration file statically and dynamically.

Monitoring the status of datanodes is done by the function call PQPing() API provided by libpqfunction which is called by pingNode() function in utils.c.

Nevertheless, CM server may become a bottleneck when MPPDB scales to thousands of nodes, largely due to the frequent datanode HA arbitration which is totally decided by the CM server.

2.1.2 Limitations of Log-shipping

The concept of write-ahead log (WAL) has been widely used in most distributed database systems. It ensures the data integrity in a way that all the changes to the database must be recorded to the log file before it's actually applied to the database. So if the node crashes it can replay the log to bring everything back to the point before the failure.

Log shipping is used in GaussDB for WAL replication in order to migrate database to reduce the system downtime. By archiving the logs continuously and dump it to the standby nodes periodically, the primary node can sync its current state with other cohorts in the same view to balance its workload as well as reduce the chance of downtime. While queries can reach the hot standby, a data loss problem may arise with a primary failure before the committed log replayed serially to the backups in the asynchronous commit. The lag window is addressed especially when the primary encounters severe loading with updates before being down. All the transactions that occurred after the last backup will be lost forever.

2.2 Raft Consensus Protocol

2.2.1 Leadership Election

In contrast to CM arbitration in GaussDB, the backups monitor the running status of the primary in Raft. It is assumed in Raft that at any time there is at most one leader existing in a view. The leader is supposed to broadcast its heartbeats regularly. The system runs into the leadership elections state should the leader become unavailable due to disk failure or network partition. Without being notified the liveliness of the primary, the standby tries to switch to candidate at the same time send out voting request to other cohorts because of an election timeout. Once the candidate collects a majority of positive votes from its peers, it steps up to the primary. An anomaly exists in the case of split voting. When two or more standbys put themselves up simultaneously, the voting competition came to an impasse with both sides failing to hear from a majority, which process may be repeated infinitely assuming the same timeout elapses. To prevent this

kind of live-lock, the election timeout is randomized to reduce the probability of initializing leadership election from two candidates at the same time.

The use of Raft view change automates the view change and ensures that the system is accessible as long as a majority of cohorts are active. With leadership election applied, not only the loading to the CM server is lightened, but also the recovery of instances become quickly and the data-note availability was increased.

2.2.2 Log Replication

Raft provides an alternative method for log shipping. Raft uses a simple log format to describe the algorithm. Each log entry is stored along with a term number in which the entry was received by the leader and an index number identifying its position in the log. Once a leader is elected, the client requests are forwarded to the leader to be serviced. The leader commits the requests on its own log as well as on the logs of all the followers using RPC. The leader returns the result to the client if it receives acknowledges from all the followers. If followers crash or run slowly, or if network packets are lost, the leader retries to commit indefinitely, even after it has responded to the client, until all followers eventually store all log entries. A log entry is committed once the leader that created the entry has replicated it on a majority of the servers. During normal operation, the logs of the leader and followers stay consistent, so the consistency check never fails.

3 Using RUDP transport protocol in Raft

3.1 Why not TCP or UDP?

3.1.1 TCP

TCP is connection-based protocol, using three-way handshaking to guarantee the transmission reliability, so it complicates the data transmission from the beginning. During the streaming, data is transmitted in a strict order. In case of network delay or packet loss, all the subsequent data transmission will be blocked until the missing packets are recognized. This situation is especially serious among the network with higher delay and more hops, for example, in a wireless network that suffers a lot of package loss, lower bandwidth and more frequent delay. Also, for massive data transmission, the window size mechanism leads to the increasing waiting time for the ACKs from the receivers.

3.1.2 UDP

Unlike TCP, UDP doesn't need handshaking mechanism to establish the connection before transmitting data, nor does it need the ACK for successful transmission. But in consequence of that, UDP can't guarantee the transmission reliability. If a packet is lost, UDP just ignores that packet and won't retransmit it. When the datagram duplication happens, it won't detect it.

3.2 Features of RUDP Protocol

To work well with the presence of high network latency and occasional packet loss on the Internet with TCP and improve the reliability by utilizing the UDP efficiency, RUDP extends UDP by means of the following features:

(1) **Acknowledgment of receiving packets**;

(2) **Retransmission** of lost packets: Retransmission occurs as a result of receiving an EACK segment of the time-out of the retransmission timer. When a retransmission time-out occurs, all messages on the unacknowledged sent queue are retransmitted.

(3) **Resending** of lost messages: when a whole message is missed, then it will be segmented and retransmitted.

(4) Windowing and **Flow control**: the size of the receiver's input queue is configured accordingly. The recommended value of the receiver input queue size is 32 packets. The input queue size acts as a flow control mechanism.

(5) **Error Detection**: checksum is used to verify the packet correctness and integrity.

(6) **Message recovery**: different parity ratio is defined to ensure that a certain packet can be recovered without the need to be retransmitted.

(7) Protect from **packet duplication**: it will just throw away any packets received for more than once.

(8) **Allows out-of-order arrival**: fhdr is ahead of others, and will be set in the right place.

RUDP is applied to get good performance in the presence of high latencies. The main one is to make sure enough packets are kept “in flight”.

3.3 APIs provided in RUDP

Interfaces	Description
rudp_socket	Instantiate a rudp_socket to associate with a socket number
rudp_sendmsg	It will call _rudp_sendmsg, using msg_name as the destination
rudp_send_completion_poll	Send Buffer would be freed if the sending is complete and can be reused
rudp_recvmsg	Receive message and check its integrity
rudp_recv_complete	Free the msg_iov and check if each fragment has been received
rudp_close	Close the transmission when it’s safe

3.4 Attempt to apply RUDP in Raft

(1) Bind the socket number to rudp socket format

```
if((err = rudp_socket(sock, NULL)))
  {
    printf("svr/main: rudp_socket err %d\n", err);
    exit(EXIT_FAILURE);
  }
```

(2) Replace the **recvfrom** API with **rudp_recvmsg**

```
// encapsulate the message information in the message header
mh.msg_name = (const struct sockaddr *) &sockaddr_hb_from;
mh->msg_namelen = fromlen;
mh->msg_iovlen = 1;
mh->msg_iov = (struct iovec *) malloc(sizeof(struct iovec));
mh->msg_iov[0].iov_base = hbMsg;
mh->msg_iov[0].iov_len = len;
mh->msg_control = 0;
mh->msg_controllen = 0;
mh->msg_flags = 0;
n = rudp_recvmsg(vs_heartbeatSock, &mh, 0, &hbmf->handle);

// replace recvfrom API with rudp_recvmsg
//n = recvfrom(vs_heartbeatSock, &hbMsg, len,0,(struct sockaddr *)&sockaddr_hb_from, &fromlen);
```

(3) Replace the **sendto** API with **rudp_sendmsg**

```
//n = sendto(vc_heartbeatSock[i],hbMsg,len,0,(const struct sockaddr *)&vc_sockaddr_hb[i],len_sock_addr);

// encapsulate the message information into the message header
mh->msg_name = (const struct sockaddr *)&vc_sockaddr_hb[i];
mh->msg_namelen = len_sock_addr;
mh->msg_iovlen = 1;
mh->msg_iov = (struct iovec *) malloc(sizeof(struct iovec));
mh->msg_iovlen = 1;
mh->msg_iov[0].iov_base = hbMsg;
mh->msg_iov[0].iov_len = len;
mh->msg_control = 0;
mh->msg_controllen = 0;
mh->msg_control = 0;
mh->msg_controllen = 0;
mh->msg_flags = 0;

//replace sendto API with rudp_sendmsg
n = rudp_sendmsg(vc_heartbeatSock[i], mh, &hbmf->handle);
```

3.5 Concepts Reference

Key words	**Description**
TCP window	The amount of unacknowledged data remaining in the network.
RTT	Round trip time of each data packet.
Client's Receive window (servers's send window)	The number of bytes the client it is willing to receive from the server
Server's Receive window (client's send window)	The number of bytes of data the server is going to take from the client
Window size	The amount of data can be hold in a receive buffer at a time
Acknowledgment	the reply indicting the successful arrival of the data
Flow control	The method for status communication

4 References

[1] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” In USENIX Annual Technical Conference.
[2] B. Liskov and J. Cowling, “Viewstamped Replication revisited,” MIT, Tech. Rep. MIT-CSAIL-TR-2012-021, Jul. 2012.
[3] Raft consensus algorithm website. http://raftconsensus.github.io.
[4] Raft user study. http://ramcloud.stanford. edu/˜ongaro/userstudy/.
[5] Koichi Suzuki and Masataka Saito, “Postgres-XC Concept, Implementation and Achievements,” Sep. 2014.
[6] “Gauss MPP DB Architecture Logical Design Specification,” Nov. 2013
[7] Reliable User Datagram Protocol. https://en.wikipedia.org/wiki/Reliable_User_Datagram_Protocol
[8] Reliable UDP Protocol. https://www.ietf.org/proceedings/44/I-D/draft-ietf-sigtran-reliable-udp-00.txt
[9] Abhilash Thammadi, “Reliable User Datagram (RUDP)”, 2009. http://krex.kstate.edu/dspace/bitstream/handle/2097/11984/AbhilashThammadi2011.pdf?sequence=5&isAllowed=y.