MEP 17 -- Support handoff and load balance segment on query nodes

Current state: Under Discussion

ISSUE: https://github.com/milvus-io/milvus/issues/9481

PRs:

Keywords: load balance, handoff, query node, query coord, sealed segment

Released:

Summary

This article mainly introduces the importance of handoff and load balance segment functions, and proposes a solution to handoff and load balance segment without interrupting the query.

Motivation

The query node of milvus currently has the following issues:

1. In the load collection/partition process, query nodes not only load the sealed segments, but also watch dmChannels so that they can receive vectors inserted in real time, and if new vectors are continuously inserted after the load collection operation, the query nodes will create a large number of growing segments, which can only service query requests in a near-brutal manner. At the same time, data node may have flushed these vectors into persistent storage and created the vector index.
2. In the process of dynamic expansion and contraction of query nodes, there may be an imbalance in the number of sealed segments on different nodes. This results in a relatively slow query for high-load query node, which will becomes the final bottleneck for queries.

So we want to do the following

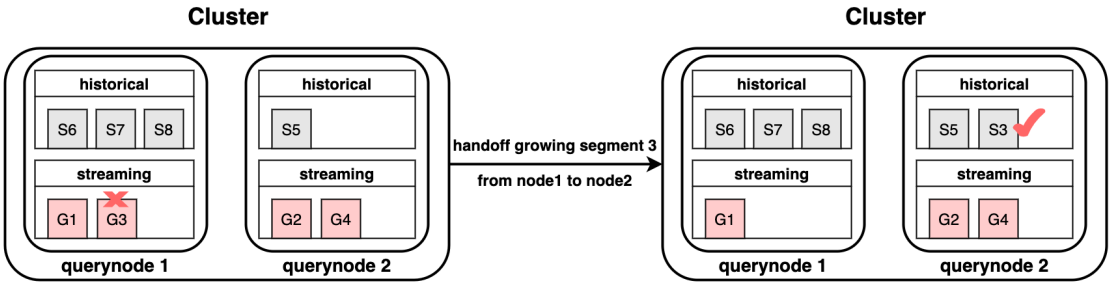
1. handoffAfter datanode has flushed segment to persistent storage, query node will load the segment's binlogs from the persistent storage and replaces the growing segment with sealed segment.
2. load balanceAutomatically balance all sealed segments on query nodes to avoid idle computing and storage resourcesand speed up the overall query.

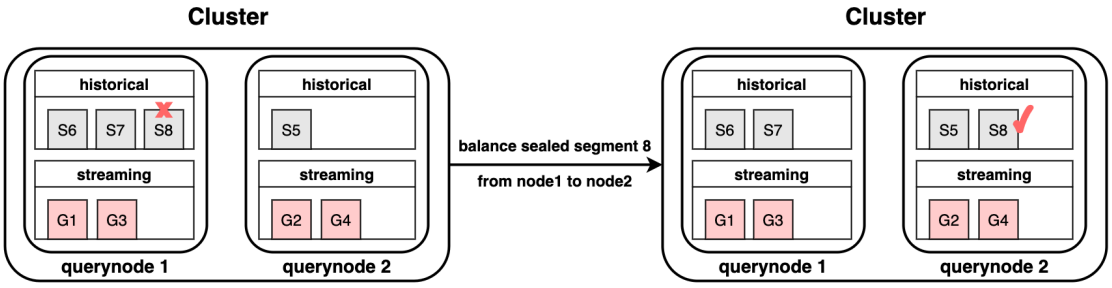
Public Interfaces

Handoff and load balance are transparent to users.

Design Details

Query node logically manages the sealed segments and the growing segments in two parts, historical and streaming module. As shown in the following figure, G3 represents the growing segment 3, and S8 represents the sealed segment 8.





Based on the diagram above, it can be concluded that the handoff and load balance process consists of two main steps:

1. Load the sealed segment onto a query node
2. Remove the corresponding sealed segment or growing segment from a query node

However, there are several issues to be considered in the implementation:

1. What if some query node associated with the handoff or load balance process is suddenly down
2. If there is a sudden downtime on query coord, how to properly resume the handoff and load balance tasks after the restart
3. If the same sealed segment exists on different query nodes at the same time, whether it will have an impact on the final query results
4. Different query nodes may not process query messages at the same speed, and how to ensure that different query nodes have the same global sealed segmentIDs when processing the same query message. If not handled well, it may cause the query to time out

Based on the above considerations, we propose the following designs:

Prerequisite: The cache meta of querycoord records which sealed segments and growing segments are loaded on each query node, and which dmchannels are watched by the query node. These metas are stored in etcd at the same time.

1.First of all, querycoord automatically generates handoff tasks and load balance tasks and writes them into etcd, and then clears them from etcd until the task is completed, ensuring that querycoord can accurately restore the task after restarting.

2. Next, query coord uses its own allocation strategy to determine which querynode the sealed segment of balance or handoff should be allocated to. The allocation strategy considers whether the total size of segments on each query node is balanced, and whether the segments of each collection are evenly distributed across all querynode

3.Querycoord directly loads the allocated sealed segments to the corresponding query node, and then the same segment will exist on different querynodes. We de-duplicate the query result based on the primary key during local reduce and global reduce to avoid affecting the query result.

4.After query coord successfully loads sealed segments on querynode, update the sealed segments list of each querynode in cache meta. For example, after balancing sealed segment 8 from node 1 to node 2, the meta of node1 changes from {S6, S7,S8} to { S6, S7}, node2's meta is changed from {S5} to {S5, S8}. If the query node suddenly goes down, directly follow the meta records of the coord to recover the query node. While querycoord updates the meta, it sends a change info of sealed segments to querychannel. The proto of sealed segments change info is as follows:

```
message SealedSegmentsChangeInfo {
 common.MsgBase base = 1;
 int64 online_nodeID = 2;
 repeated SegmentInfo online_segments = 3;
 int64 offline_nodeID = 4;
 repeated SegmentInfo offline_segments = 5;
}
```

5.Query coord updates querychannelInfo in meta, mainly updating two content, one is global sealed segments, for example, after handoff growing segment S3, global sealed segments change from {S5, S6, S7, S8} to {S3, S5, S6 , S7, S8}, the other is seek_position, that is, the position of the previous SealedSegmentsChangeInfo in the querychannel. Each query node will receive these two initialization parameters when the querychannel is watched. This ensures that when the specified query message is processed , The global sealed segments received by all querynodes are consistent.

```
message QueryChannelInfo {
 int64 collectionID = 1;
 string query_channelID = 2;
 string query_result_channelID = 3;
 repeated SegmentInfo global_sealed_segments = 4;
 internal.MsgPosition seek_position = 5;
}
```

6.After querynode consumes SealedSegmentsChangeInfo from querychannel, it will do the following three things:

- Update the list of gloabal sealed segments in the query collection according to online_segments and offline_segments in SealedSegmentsChangeInfo

- If the growing segment ID in streaming appears in the list of global sealed segments, clear the corresponding growing segments from streaming module
- If the current nodeID==offline_nodeID, clear the sealed segments in offline_segments from historical module

Test Plan

- Unit tests
- CI tests