Combined placement for Join Efficiency in Gauss MPPDB

Chunli Yu, Data Management Engineering Intern
R&D Software Lab, Huawei Technologies

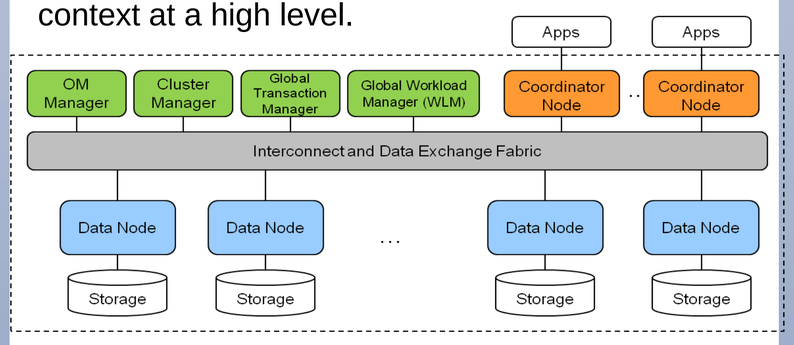
Abstract

The need for scalable, high-performance data-stores has led to the development of Gauss MPPDB, which achieves scalability by partitioning data into evenly distributed parts. This way a database query can be executed in parallel with a much smaller data set. However, when join is involved, especially non-collocated join, MPP DB needs data communications among DNs. Especially when cluster size is large, the data shuffle cost can be extremely expensive.

This project suggests that there is no need to compromise join efficiency for scalability. A hybrid data store is addressed that enables efficient join query by rethinking data placement with replication for reference table as well as hash distribution. Experimental results from multiple dimensions shows that this hybrid placement method achieves a significant performance gain.

Introduction

An MPP database usually splits a data set into smaller parts on to different physical devices evenly. Thus, it heavily relies on the data exchange among data nodes. The following diagram describes the overall logical system context at a high level.



We aim to find a balanced way for data store to counterweigh the busy shuffling traffic coming with non-collocated joins between the Data Nodes listed above. The tuning is based on a model formula:

$$Performance\ gain = \frac{\sum_{i=1}^{n}(\alpha_i) + \mu * n^2 - \sum_{i=1}^{n}(\beta_i)}{\sum_{i=1}^{n}(\beta_i)}$$

Where n is the cluster size, μ is the traffic latency, α and β are the hash join and combined join overhead in each data node respectively.

Methods

- Compare traditional hash joins based on distributed key and non-distributed key.
- Compare hash-to-hash join to hybrid join with an increasing data size on a fixed-size cluster.
- Compare hash-to-hash join to hash-to-replication join with fixed-size dataset on an expanding cluster.
- Evaluate the system usage, like CPU, I/O, Memory and network.

Setup

The experiments were conducted on SUSE Linux hosts featuring Intel® Xeon® E5-2620 CPUS (2.0GHz, 15.3Mb Cache, 6 cores per CPU) and 3.3T sdb on the platform of yellow zone - USRD Cloud. I will use the datasets ranging from 1GB to 69GB generated by the dbgen testsuit.

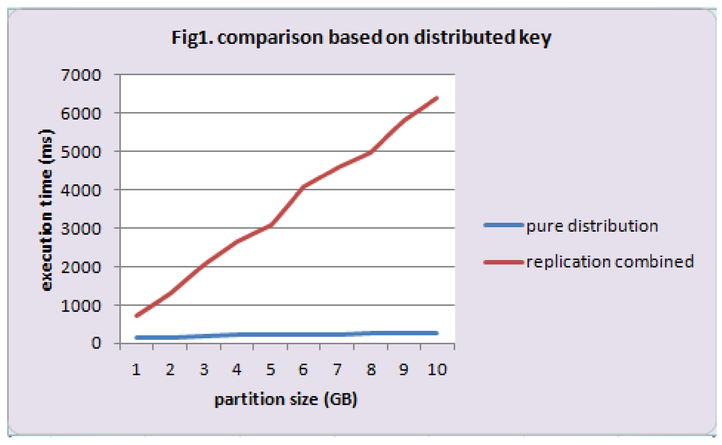
Results

Execution Pof

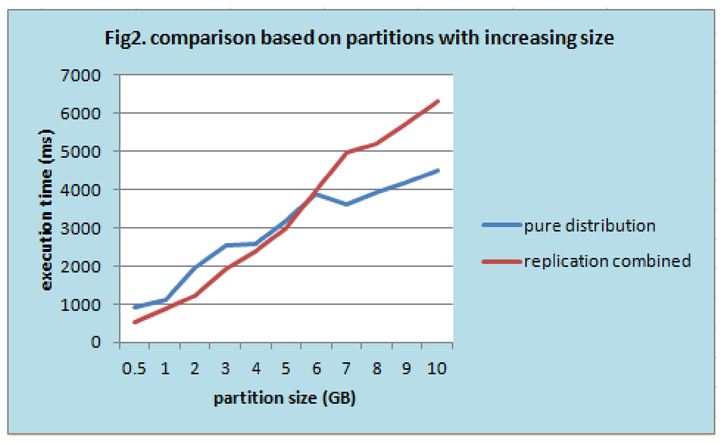
```
|---------------------- query plan on pure hash placement ------------------|
Row Adapter
 -> Vector Aggregate
    -> Vector Streaming (type: GATHER)
       Node/s: All datanodes
       -> Vector Aggregate
          -> Vector Hash Join
             Hash Cond: (partsupp.ps_suppkey = lineitem_dis_05x.l_suppkey)
             -> Vector Streaming(type: REDISTRIBUTE)
                Spawn on: All datanodes
                ->  CStore Scan on partsupp
             -> Vector Streaming(type: REDISTRIBUTE)
                Spawn on: All datanodes
                ->  CStore Scan on lineitem_dis_05x
|-------------------------------------------------------------------------|

|-------------query plan on hash placement and replication placement-------|
Row Adapter
   -> Vector Aggregate
      -> Vector Streaming (type: GATHER)
         Node/s: All datanodes
         -> Vector Aggregate
            -> Vector Hash Join
               Hash Cond: (partsupp.ps_suppkey = lineitem_rep_05x.l_suppkey)
               -> CStore Scan on partsupp
               -> CStore Scan on lineitem_rep_05x
|-------------------------------------------------------------------------|
```

The above query plans show that when it refers to the partition on another data-nodes during joining, the cluster needs to redistribute the shards and spawn on all data-nodes.

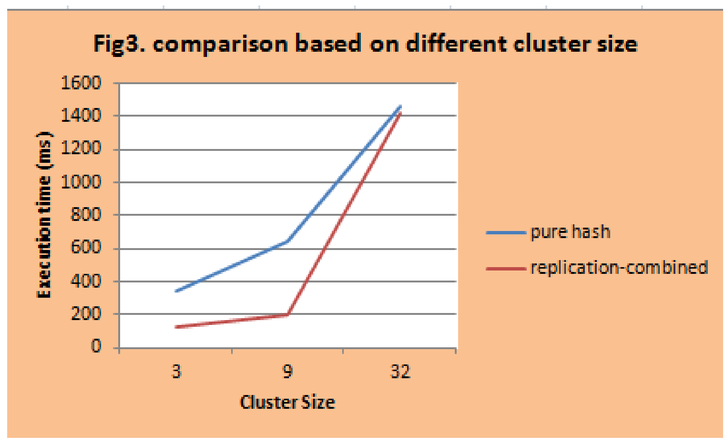
Performance Evaluation



- If the join is based on the distributed key, then execution is much faster in which case the push hash placement should be applied.

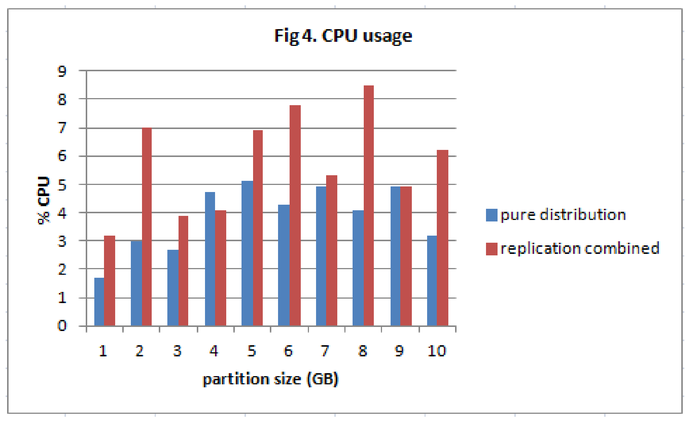


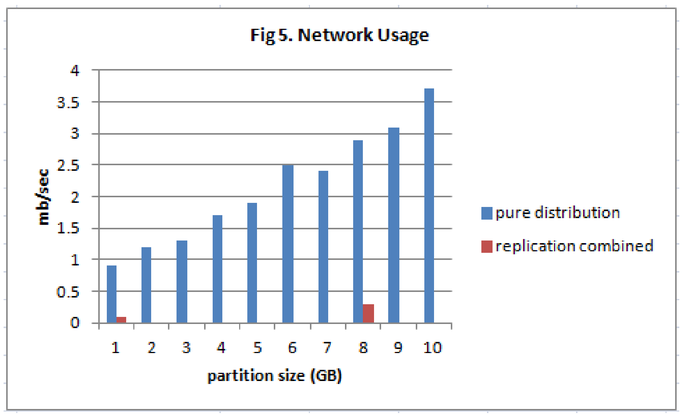
- If one of the hash placement is replaced by replication when the partition size is reasonable, the replication combined method can achieve better performance gain.



- if the cluster is small, it's hard to recognize the superiority of the replication combined strategy . However, when the cluster is expanding greatly, the execution time soared upward.

System Performance





From the above diagrams, we can see that with the replication combined method, the cpu usage exceeds that of the pure hash method. Besides, Apparently, the biggest advantage for join among highly distributed storage system is the network communication, which is shown in Figure 5.

Conclusions

Gauss MPPDB provides a supreme distributed data-storing framework, but its design posed challenges to attain the best performance in join execution due to tightly coupled shuffles.

In this poster, balanced combining placement is proposed to minimize data shuffling. It proactively analyzes table size and execution plans, then distributes data by partition or replication accordingly. This project demonstrated not only that the quicker execution time is possible, the system performance is also better.

References

1. Huawei technologies Co., LTD. (2013). "Gauss MPP DB Architecture Logical Design Specification".
2. Postgres-xc pgbench: http://postgres-xc.sourceforge.net/docs/1_1/pgbench.html.
3. Koichi Suzuki. (2012). "High Availability in Postgres-XC".
4. M. O. Akinde1 , M. H. Bohlen1 , T. Johnson2. "Efficient OLAP Query Processing in Distributed Data Warehouses".

Name: Chunli Yu
Team: Database Team
Mentor: Le Cai
Email: cyu22@syr.edu