HRDBMS: A NewSQL Database for Analytics

Jason Arnold, Boris Glavic, Ioan Raicu
Department of Computer Science
IIT
Chicago, USA
jarnold6@hawk.iit.edu, bglavic@iit.edu, iraicu@cs.iit.edu

***Abstract*— HRDBMS is a new type of distributed relational database that uses a hybrid model by combining the best of traditional distributed relational databases with more modern workflow/framework based relational databases. This allows HRDBMS to take advantage of years worth of research regarding query optimization, while also taking advantage of the scalability of workflow-based systems. Furthermore, it uses a customized execution framework that removes the performance challenges that have been observed with running SQL on Map/Reduce and Spark. These include materialization of intermediate results, lack of a global cost-based optimizer, unnecessary sorting, lack of good index support, lack of good statistics, lack of full DML support, and the large number of map and reduce phases that are required.**

Keywords—SQL, analytics, distributed query processing, relational databases

I. BACKGROUND

The increasing scale of data to be processed for analytics has been a problem for the IT industry for years. Many types of solutions to this problem have been proposed. First MPP relational databases were proposed. These used a shared-nothing architecture to try to parallelize the query processing across multiple nodes. While this proved to be effective at very small numbers of nodes, this approach did not scale to even medium-sized clusters. Second columnar databases were proposed. For many types of analytical queries, these proved more efficient than their row-based counterparts. But, they suffered the same scalability issues that standard MPP relational databases faced. Most recently, we have seen a number of workflow/framework based SQL engines. This includes things like Hive, Shark, Spark SQL, and Dremel. While these types of approaches scale much better than traditional database approaches, they tend to perform very poorly because the database has to reduce the queries into a DAG of jobs to be performed on the underlying workflow framework. HRDBMS is an attempt to unify the best things about workflow-based SQL engines with the best things about traditional MPP relational databases.

II. HIGH LEVEL ARCHITECTURE

An HRDBMS cluster is broken into 2 types of nodes. There are coordinator nodes and worker nodes. Coordinator nodes only hold system metadata and are responsible for query planning. Worker nodes only hold user data and are responsible for the majority of query execution. Queries enter the system by a client submitting SQL to a coordinator node. The coordinator node then runs the query through the optimizer, which uses statistics which are stored in system tables. The output of the optimizer is a query plan, which then gets submitted for execution. Query results are eventually fed back up to the coordinator node that planned the query. Final sorts or aggregations may occur on the coordinator node, if they are small enough. The coordinator node then forwards the result set on to the client.

The optimizer is a cost-based optimizer as with traditional relational databases. The benefits of cost-based optimization are well established. Secondly, it uses an execution framework modeled after Map/Reduce, and the scalability of such frameworks is also well-established. Third, HRDBMS varies from the typical Map/Reduce model in ways that improve the performance of SQL queries. These differences are outlined in section III.

Unlike Hive, HRDBMS also supports INSERT/UPDATE/DELETE operations with full transactional consistency (ACID compliance).

III. HRDBMS'S DISTRIBUTED EXECUTION ENGINE

Queries execute in the HRDBMS framework. The HRDBMS framework is modeled after Map/Reduce, but contains many improvements and customizations specific to executing relational queries.

We made the following observations about Map/Reduce that we believe lead to the slowness of SQL operations for databases such as Hive. First of all, a reduce phase is really just a special type of map phase. It again emits key/value pairs. The difference is that these key value pairs do not go through a shuffle and instead are directly written to disk. It takes another (basically no-op) map phase to shuffle the output from a previous reduce phase so that data is correctly co-located for processing. In HRDBMS, we therefore forego the use of reduce phases and string a series of map phases together with shuffles in between them. Any of these map phases can perform any type of relational operation, including aggregation. This also eliminates the write to disk that occurs in Map/Reduce at the end of each reduce phase. HRDBMS only materializes the data when necessary. This also significantly improves performance as noted by others [1].

The first map phase will read the necessary input data from disk. The data read may either come from a table or from an

index. Data is not stored on HDFS. Instead the data is stored directly on the local filesystem of each node. Coordinator nodes are aware of how the data is partitioned across each of the worker nodes. Worker nodes may each contain a large number of physical disks. The coordinators are also aware of how the data is partitioned across each disk on the worker nodes. The first map phases are always scheduled to run on the node where the data they read resides. It is not a preference for placement of the map task like it is in Map/Reduce. Data locality is guaranteed. These map tasks can then perform any type of relational operations needed, and eventually they do a mapping of each row and write it out to the shuffle.

It's worth pointing out a few more differences between the HRDBMS framework and Map/Reduce. First of all a HRDBMS shuffle does not guarantee an ordering of the key values that the next map phase receives. It just guarantees that all the rows with the same key are sent to the same node for the next phase of processing. This is sufficient to implement distributed joins or aggregation, and reduces overhead be eliminating unnecessary sorts.

Also, these map tasks are inherently parallel. Each map task is defined in terms of the I/O and relational operations it will perform. HRDBMS is designed such that it is easy to construct map tasks which perform different relational operations in parallel, do I/O in parallel, and even use parallelism within the execution of a single relational operator. For example, table data is not only partitioned across nodes, but also partitioned across disk drives on each node. A separate I/O thread is assigned to each disk.

Another performance improvement comes from the combination of not materializing intermediate files on disk, and not sorting during the shuffle. These changes allow 1 map phase to start processing data before a previous map phase finishes running. In fact for some queries, all of the map phases involved in the query may immediately begin to do work when the workflow begins executing.

Eventually, the map tasks all map their rows back to the coordinator node that planned the query. The coordinator then returns the result set back to the client.

IV. HRDBMS OPTIMIZER

HRDBMS takes advantage of years worth of research into traditional relational database SQL optimization to build efficient workflows for execution. Query planning starts out very similar to query planning for a traditional relational database. Standard transformations such as operator reordering are applied, join types are chosen, cardinalities are estimated from statistics, selection and projection are pushed down, etc…

In HRDBMS, the cost of a query is dominated by the cost of the shuffles. Therefore, the optimizer attempts to do 2 things. First it attempts to reduce the number of shuffles by taking advantage of the way the data is partitioned on disk as much as possible. Secondly, it attempts to reduce the number of rows that are passed through each of the shuffles. This is a similar process to join enumeration in a standard relational database.

The optimizer then adds combiner steps where they can be used. Again, this reduces the amount of data that has to pass through the shuffle. These combiner steps are just like in Map/Reduce where they are local to 1 node. However, HRDBMS also prefers multiple smaller parallel shuffles to one big shuffle. So, the optimizer may also choose to insert "combiner-like" steps to do pre-aggregation, where each combiner-like step processes data from a subset of the overall set of map tasks. There may be more than 1 round of these combiner-like tasks depending on data cardinality and the number of nodes involved.. The optimizer may also choose to do the same sort of thing for sorting. This is essentially multi-pass k-way mergesort, but with parallelism on each of the passes. The optimizer will insert these combiner-like steps in an effort to reduce the number of neighbors that each node must communicate with. Tests have shown that this results in more efficient network communications.

Lastly, the optimizer decides where to replace table scans with index access. The final workflow is composed and it is submitted to the HRDBMS framework for execution.

V. RESULTS

We have run micro-benchmarks as well as the TPC-H benchmark at the 100GB scale on clusters ranging from 4 to 32 nodes. All nodes are Amazon EC2 m3.2xlarge instances. HRDBMS was compared against Hive and DB2 (an enterprise class traditional distributed relational database). In micro-benchmarks, HRDBMS outperformed both Hive and DB2. In the TPC-H benchmark, HRDBMS was a few percent slower than DB2, but several times faster than Hive. HRDBMS did show a higher speedup from 4-32 nodes in TPC-H testing than did DB2.

VI. FUTURE WORK

Fault tolerance is not yet implemented in HRDBMS, but the system is already rack-aware and aware of how data is partitioned across nodes and disks. The plan is to have HRDBMS maintain on-rack and off-rack replicas that can be used in the event of primary node failure. This could be extended to preferring secondary copies of data when certain nodes have more capacity at current than other nodes. Data modifications affecting nodes that are down would be placed into pending-work queues. The work in these queues must be completed and successfully committed before a node is allowed to rejoin the cluster.

REFERENCES

[1] R. Xin, J. Rosen, M. Zaharia, et al, "Shark: SQL and rich analytics at scale," Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 13-24, 2013