MammothDB

MammothDB Overview

What is MammothDB?

MammothDB is an SQL-compliant RDBMS on steroids, for building and running interactive analytical enterprise applications on a cluster of commodity computers.

A quick snapshot, MammothDB is a data warehousing infrastructure that is:

- Scalable
- Fast
- Affordable
- SQL compliant

A longer view, MammothDB is:
An inexpensive alternative to existing shared-nothing RDBMS solutions (such as Teradata) for large-scale enterprise data warehousing and analytics. Its mission is to lower significantly the investment hurdle for companies that want to start a DWH/Business Intelligence program. Combining commodity hardware and network components, along with a simple approach to distributing data and query processing, MammothDB, helps keep both R&D and implementation costs low. Cluster deployment allows the solution to scale smoothly, and in small increments, simply by adding hardware as the data size grows from relatively small (100s of GB) to Big Data scale.

Even though MammothDB uses Apache Hadoop infrastructure for distributing data and computation, it does not share the Big Data philosophy of 'ingest whatever data as it comes at high velocity, without schema or structure. Let the developer figure out the structure at query time'. This gives MammothDB a distinct edge for handling enterprise data, compared to SQL products running on Big Data infrastructure like Hive, Hadapt, Impala, PrestoDB. It is efficient, interactive, and fully interoperable with ETL and reporting tools that expect sub-second response (e.g. Cognos, Microstrategy, Tableau, etc).

How MammothDB Works – High Level

MammothDB has implemented our vision for the future of Tough Data. What we call "Tough Data" is enterprise data coming from a company's line-of-business systems that is sufficiently big or complex to make open-source database products choke, yet it is not web-scale. We have taken a simplistic approach to scale out those open-source database products so that they can handle the large volume and complexity at a fraction of the cost of running a commercial data warehousing product.

MammothDB

In order to achieve interactive response times (yes, single digit milliseconds!), MammothDB stores the data in 'data engines' sitting on the nodes in the cluster, pre-structured for optimal query execution, thus loading local node CPU/memory and disk and minimizing the need for data movement across the network. This is substantially different from most current production Big Data frameworks, which store data as flat files in a distributed file system and, at query time, process and move massive amounts of data.

This is the principle of operation: when data is being loaded from an external application, MammothDB automatically distributes table rows around the cluster. Each cluster node gets a share of the overall data, and stores it locally in an optimized columnar format.

At query time, MammothDB accepts a standard SQL DML statement, and has each node run parts of it on the locally stored data, producing a partial result. Partial results are then aggregated through Hardtop's Map-Reduce interface and are returned back to the client.

Does it integrate with anything?

Yes, it speaks SQL!

And it's not some remote academic or proprietary dialect of SQL, either! MammothDB speaks a dialect that popular reporting, analytical and data mining tools understand. So if you want to analyze your data in a Microsoft Excel pivot table by dragging and dropping fields, you're good to go. We've run Microsoft SSAS and SSRS, IBM Cognos, Microstrategy, and Penaho Reporting and Analysis using MammothDB as the backend.

What MammothDB is not?

It's not a transactional processing system.

Although there is transactional support for basic INSERT/UPDATE/DELETE operations in MammothDB, it comes with a performance penalty when dealing with transactions affecting one or a few records. The role of transactional support in MammothDB is to facilitate integrations with ETL tools like MS SSIS, ODI, Informatica, Pentaho Data Integration. MammothDB has been designed for analytics and is *not* intended as a scalable platform to run millions of concurrent transactions per second.

It's not NoSQL.

MammothDB is aware it is a bit old-fashioned, but nevertheless *likes* SQL and even thinks SQL is cool when it comes to knowledgeable humans and many great reporting/analysis/charting packages running analytical queries.

We still recognize that some **NoSQL** tools are extremely good, much better than SQL-based systems, for transactional environments supporting millions of concurrent updates coming from millions of web users. That's why we like to read NoSQL as Not Only SQL – as in "why not use the best tool for the job, run your line-of-business systems on a NoSQL backend, and your analytics on an SQL backend".

It's not SQL-over-Hadoop.

SQL-on-Hadoop tools like Hive, Impala, PrestoDB allow SQL to be run over unstructured data that's already available in HDFS. Their mission is to provide an easy and familiar interface to data maintained in an infrastructure that still offers a very immature toolset. Data warehouse architects and build and run teams have to learn completely new skills and tools in order to use Hadoop and SQL-on-Hadoop for their job. By contrast, MammothDB is actually an RDBMS, it lets data warehouse architects and build and run teams build larger data warehouses and analytical applications using familiar skills and tools.

MammothDB runs on Big Data infrastructure, it integrates easily into Hadoop's analytical tools and offers the opportunity to use best-of-breed approach, which is to share the same infrastructure to

- run Big Data Tools for analyzing and uncovering patterns in web-scale unstructured data
- run SQL-based tools for enterprise data and interactive analytics

What Can I Use MammothDB For?

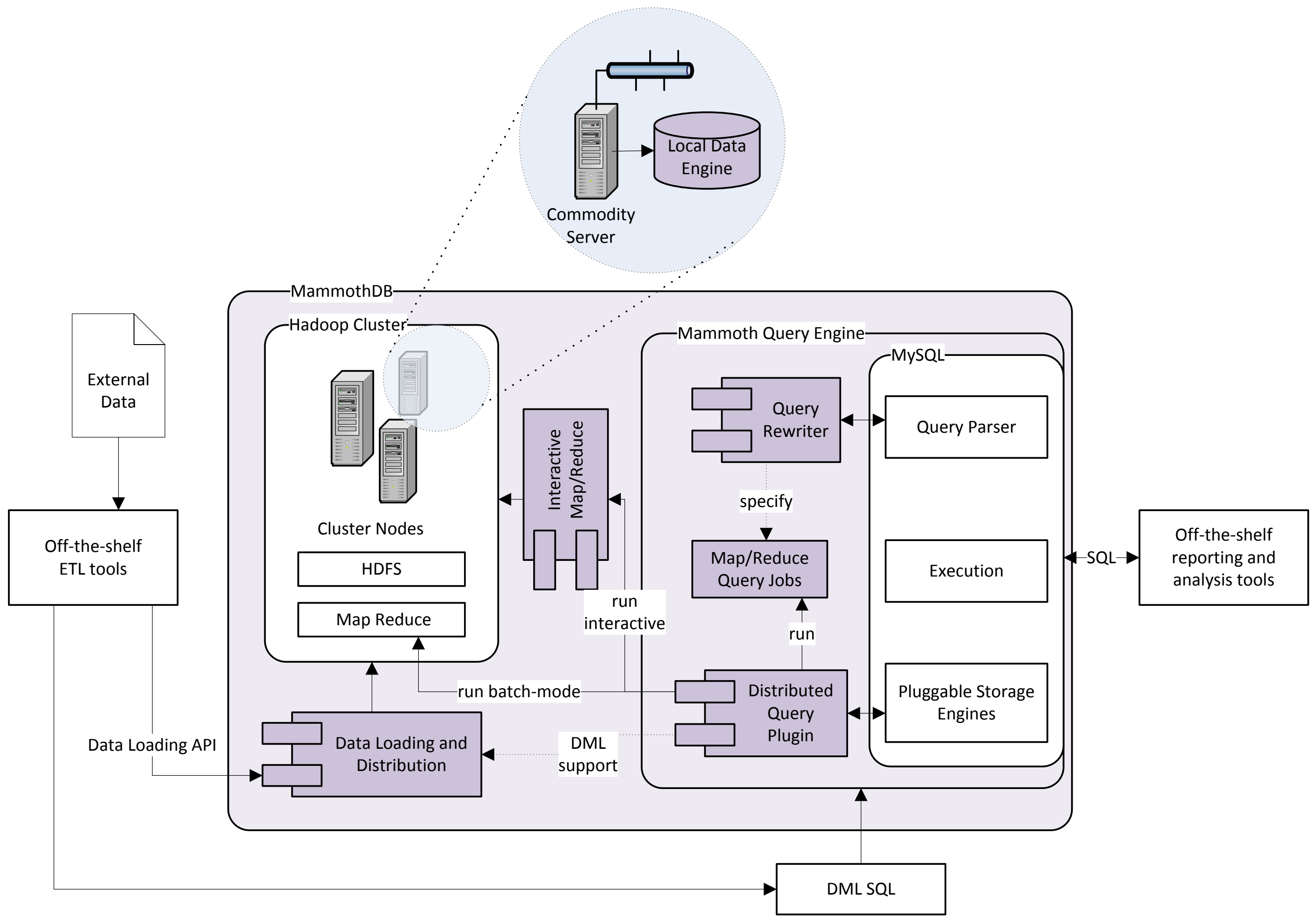
Here are a few ideas that we are sure about:

- You have lots of terabytes of data, and other SQL-based solutions to analyze it are just too expensive in terms of licenses and hardware to run. Or they wouldn't integrate with tools for reporting and analysis. MammothDB will fit right where your 'X-brand shiny uber-expensive fancy SQL Server' box sits on your architecture chart;
- Or, you want to build a BI SaaS online service, and want to offer your clients the elasticity and the benefits of sharing scaled out infrastructure;
- You might want all your corporate IT people to share powerful infrastructure to run their data warehousing initiatives; or
- You want to empower business users to analyze data that just won't fit into their favorite version of Excel (and, alas, that is just millions of records);

MammothDB

Bird's Eye View of Architecture

This is how it all works:



Firstly, data gets loaded into tables. During the load process, it is automatically distributed to the nodes in the cluster using one of these strategies:

- **Replication**: complete copies of small lookup tables (aka dimension tables in BI speak) are kept on every node;
- **Partitioning**: large tables with data points (aka fact tables) are partitioned so that every node keeps only a subset of all rows. The data distribution component makes sure (through the use of row-level hashing) that each node gets roughly the same number of rows from a partitioned table, so that no single node will have to process significantly more data than the others at query time.

Standard SQL DDL is used for creating the table structure, and the LOAD DATA command can be used to insert records from an external flat file. Alternatively, data can be loaded directly through an API.

On a node, data gets stored in a local data engine, a column-oriented RDBMS. The default local engine is Infobright Community Edition (ICE, http://www.infobright.org/), but it can be replaced by another SQL-compliant DB engine. In our lab, ICE has shown consistent compression ratios of 8:1 or better when loading data from production data warehouses and excellent response times when running star-schema queries.

The client connects to a MySQL server instance. All methods of client-side connectivity to MySQL, such as ODBC, JDBC, OLEDB and OLEDB.Net, are supported. When a SQL query arrives from the client, it is first intercepted, analyzed and re-written into:

- parts that will be run on each node against the local data;
- map-reduce specifications to aggregate intermediate results;
- parts that are run on the client-facing MySQL instance;

The map-reduce specification and the queries to run locally are stored as a virtual table in a distributed query pluggable storage engine deployed on the client-facing MySQL instance. When this virtual table is accessed by the execution engine, a map-reduce job runs on the cluster and provides back the results in tabular format.

As a short illustration, consider a query that joins a fact table with a lookup table in order to aggregate and order totals by some attribute:

```
SELECT A.attribute, SUM( M."data point" ) total
  FROM metrics M
  JOIN attributes A
    ON metrics.lookup_id = attributes.lookup_id
 GROUP BY A.attribute
 ORDER BY total DESC
```

This will be re-written in three parts:

A. <u>query to run on each node</u>

```
SELECT A.attribute, SUM( M."data point" ) total
  FROM metrics M
  JOIN attributes A
    ON metrics.lookup_id = attributes.lookup_id
 GROUP BY A.attribute
```

B. <u>map-reduce specification</u>

map: run A, output key:=attribute, value:=total

reduce: output key:=attribute, value:=sum(total)

C. <u>query to run on MySQL</u>

```
SELECT A.attribute, A.total
  FROM mammothdb.vtableA A
 ORDER BY A.total DESC
```

Parts A and B form the definition of a virtual table, vtableA in the mammothdb schema. When MySQL executes the re-written query C, it will request access to vtableA. At this point, the pluggable storage engine will run the map-reduce job. Map tasks will run the SQL statement A against the respective local data engine, and reduce tasks will aggregate the results which are then returned to the pluggable storage engine.

Depending on the expected size of the result and execution time on the local nodes, the map-reduce job will be run through one of two alternative Map/Reduce frameworks with different reliability and performance characteristics:

- Standard Hadoop Map/Reduce: for long running (>1 min) jobs producing large result sets. The standard framework ensures that the job will resume from the point of failure, with minimal loss of intermediate results, even if some tasks fail (e.g. due to component failure) during execution;
- Interactive Map/Reduce: for short running (<1 min) jobs, MammothDB provides an alternative map/reduce framework that shares the Hadoop client API, but targets minimal overhead (measured in the order of 10-20ms per job). The interactive map/reduce framework handles failure by re-starting the entire job;

The availability of an interactive map/reduce framework allows MammothDB to work smoothly with smaller data sets (in the order of 100 GB) where Hadoop tools (like Hive, Pig) show latencies in the orders of tens of seconds or even minutes. It also makes integration with reporting and analysis tools possible which typically fire tens of queries to the backend upon each user interaction, and expect responses within milliseconds.

Handling of Failure

Like other Big Data solutions, MammothDB expects nodes to fail. In order to make sure the cluster is still available and queries still run after a node goes down, redundant copies of the data are stored on multiple nodes. More specifically, partitioned tables will have each partition available on several nodes (how many depends on a system-wide parameter called 'replication factor'). Once a node goes down, other nodes will allow temporarily access to its data until an overworked IT person replaces it. Once replaced, MammothDB will move the data from already available nodes to the new ones, eventually achieving the same configuration as before the failure.