jethro

Accelerating Big Data Performance

Technical White Paper

Making Interactive BI for Big Data a Reality

Achieving truly interactive response times on Hadoop and Amazon S3

April 2015

Table of Contents

Introduction

This white paper explains how JethroData can help you achieve a truly interactive response time for BI on big data, and how the underlying technology works. It starts by analyzing the challenges of implementing indexes for big data and how JethroData indexes have solved these challenges. Then, it discusses how the JethroData design of separating compute from storage works with Hadoop and with Amazon S3. Finally, it briefly discusses some of the main features behind JethroData's performance – including I/O, query planning and execution features.

Part 1: Indexes for Big Data

Why Use Indexes?

Indexes are a fundamental performance feature of any high-performance database. Indexes accelerate selective queries by orders of magnitude, as they allow the query engine to fetch and process only relevant data, instead of scanning all the data in the tables.

In the past decade, indexes have been delegated to a minor role in high performance analytic databases, for several reasons:

➔ Maintaining standard indexes is expensive

Using standard indexes dramatically slows down data loading, especially with bigger data sources. This is true for most index implementations, including B-tree, bitmap and others.

Technically, indexes are typically divided into small blocks (e.g., every 8kb). These blocks are updated as new data is inserted – updating the list of row IDs per value. So adding new rows to a table leads to many random writes across all indexes.

As a consequence, administrators typically only index a few columns in advance. Consequently, many ad hoc queries do not benefit from indexes. Also, adding new indexes is only done after careful testing and long negotiation to minimize index impact on the data loading process.

➔ Standard indexes are not compatible with HDFS or Amazon S3

With standard indexes, adding rows to a table leads to many random updates to the index files. While standard (POSIX) file systems support random updates, modern big data storage solutions like HDFS and S3 do not. With HDFS, existing files cannot be updated – HDFS only supports appending new data at the end of an existing file (the MapR

HDFS implementation is the only exception). Amazon S3 does not support random writes (or file append) as well.

Consequently, it is impossible to implement standard indexes on HDFS or Amazon S3. As an example of this challenge, when Pivotal migrated their Greenplum database to HDFS (and renamed it HAWQ), they removed support for indexes.

➔ **Indexes don't accelerate all types of queries**

Indexes do not provide a performance advantage for non-selective queries, meaning queries that need to process a large percentage of the data (as we will see later, JethroData has a different mechanism to optimize these types of queries).

As a result, most SQL engines introduced in the last decade were designed with an MPP architecture: distributing the data across many nodes, where each node processes all of its local data for every query. It is a simple architecture that handles large aggregate queries well, but requires many servers to be effective, consumes many resources per query across the cluster (limiting SQL concurrency) and does not benefit from the dramatic performance advantage that indexes provide.

.

JethroData Indexes

JethroData indexes were specifically designed to solve the first two problems. In other words, they were designed to work natively on HDFS and Amazon S3 without the expensive index update overhead. As index maintenance is relatively cheap in JethroData, it allows every single column to be automatically indexed, thus accelerating many more queries without requiring an administrator decision for every column ("to index or not to index?").

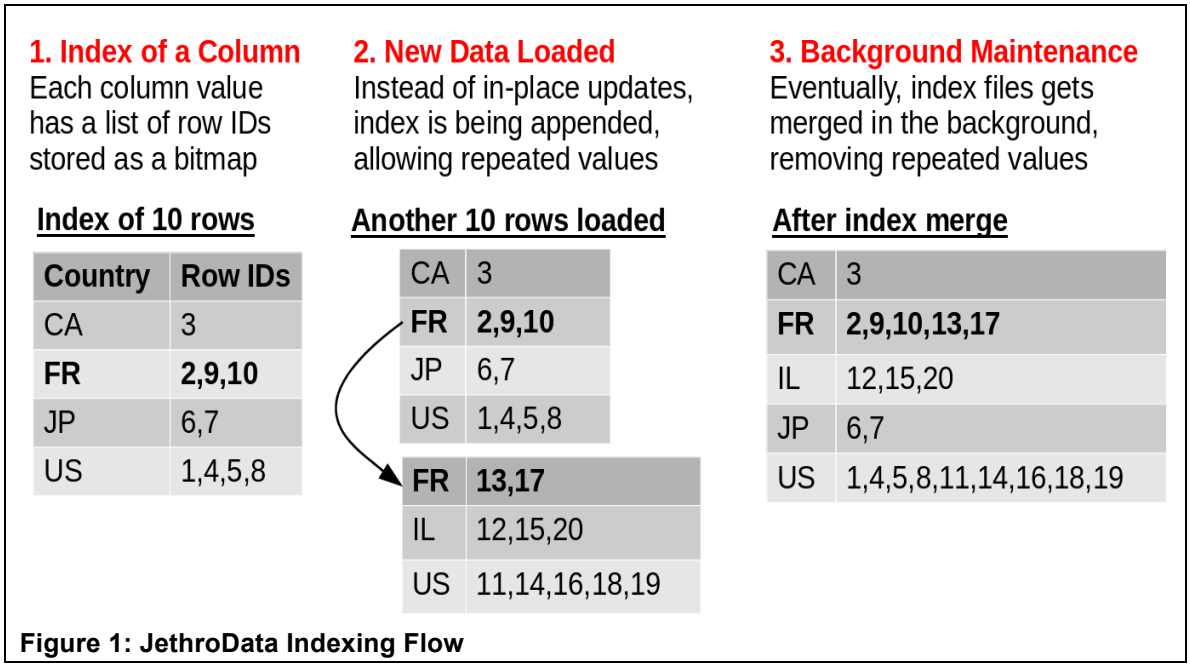

Figure 1: JethroData Indexing Flow

How is that possible?

When new rows are loaded into a table, JethroData does not go back to the existing index blocks and updates them. Instead, it adds them to the end of the index, allowing duplicate index entries.

When new data is loaded, the `JethroLoader` utility generates a large sequential write at the end an each index, instead of the conventional method of updating many index entries in-place. This leads to a dramatically lower impact of each index on the load process, compared to standard indexes.

During query execution, the `JethroServer` engine reads all the relevant index fragments. For example, if a `WHERE` clause limits results to a specific country (e.g., `WHERE country = 'FR'`), the engine may read and processes several smaller bitmap fragments, each covering a different range of rows.

At a later point, a background process (`JethroMaint`) periodically merges duplicate index entries by replacing several smaller index files with a single larger file and removing duplicate value entries. This is done in a non-blocking fashion - without affecting running queries or other data loads. It is similar in spirit to how Apache HBase performs compactions to reduce the number of files and remove duplicates and expired or deleted rows

In summary, JethroData solves the challenge of indexing at scale, automatically index all columns, and enables users to enjoy the performance of indexing without compromise.

Part 2: JethroData Architecture

The JethroData engine is installed on its own dedicated query cluster – one or more nodes. The JethroData nodes are connected to a scalable, big data storage solution – either Hadoop HDFS or Amazon S3. The JethroData query nodes themselves are stateless: all of their indexes are stored in HDFS or S3. JethroData users typically work with a BI tool, such as Qlik, Tableau or MicroStrategy, over ODBC/JDBC to enable fast, interactive BI.

In this chapter we will first discuss JethroData's architecture in the context of Hadoop and HDFS. We will then go over specific attributes of JethroData for Amazon S3.

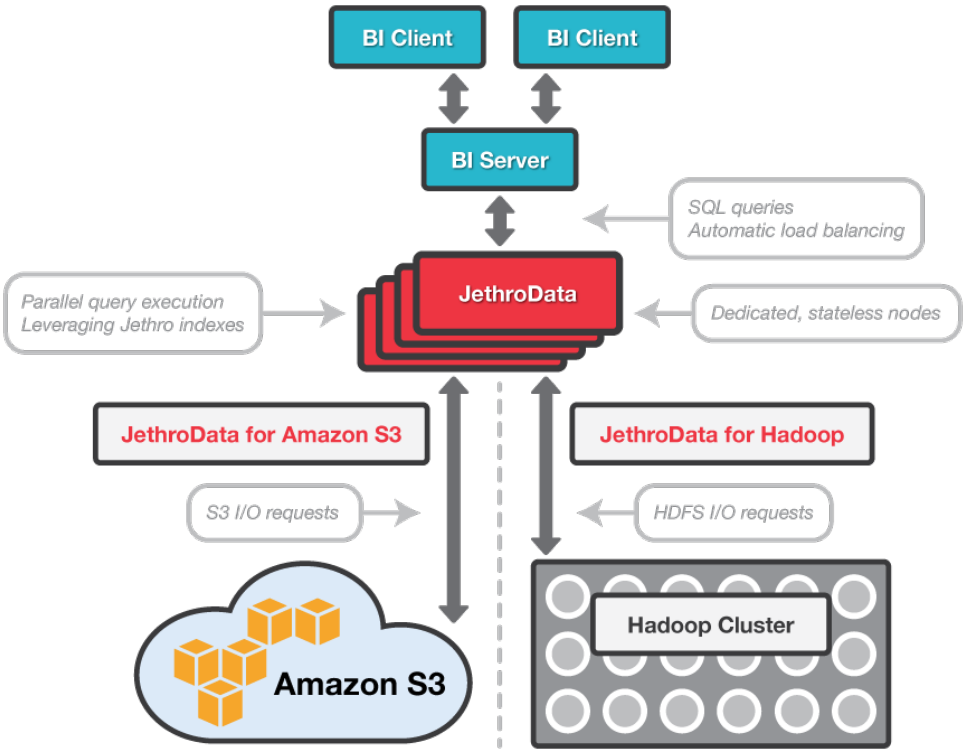

Figure 2: jethroData Architecture

JethroData for HDFS

When connected to Hadoop, JethroData is not installed directly on the Hadoop cluster nodes. Instead, it is installed on its own set of dedicated nodes next to the Hadoop cluster (sometimes called edge nodes). It does, however, store all its data on HDFS, so it works as a standard HDFS client.

This architecture has several benefits:

➔ Ease of deployment

Separating JethroData to a few dedicated nodes means that no change is required on the Hadoop cluster – no new component to install and no configuration changes. All that is required is a dedicated user with access to its own HDFS directory to store the indexes. This means it is very easy and safe to try out or deploy JethroData.

➔ Query-optimized nodes

In many Hadoop clusters, the sizing of the physical nodes is optimized either for batch processing or as a compromise between several workloads, minimizing costs. As newer, memory-optimized technologies emerge, modifying existing cluster hardware to accommodate more RAM or a local SSD is a complicated task – both effort-wise and cost-wise. With JethroData, you can easily deploy one or more higher-powered nodes to support interactive BI users, without the need to upgrade existing nodes in the cluster.

➔ Better co-existence with other workloads

Most Hadoop vendors envision Hadoop as a central computing resource that is shared across many different workloads (sometimes called an Enterprise Data Hub or Lake). However, most SQL-on-Hadoop technologies can't easily share the cluster. They are designed to consume a significant portion of memory, CPU and I/O bandwidth from every node in the cluster at once, to provide a decent response

time (*"MPP architecture*"). This leads to an inherent contention over resources amongst interactive BI workloads and other workloads in the cluster. As more users are added it becomes increasingly harder to manage. With JethroData, the query processing is done in JethroData-dedicated nodes, so that interactive BI users have a minimal impact on the Hadoop cluster (just some HDFS accesses). This enables a true sharing of the cluster between interactive BI users and batch workloads.

➔ Easy scale-out for concurrency

JethroData query nodes are stateless. As you add more concurrent users, you can easily start additional JethroData nodes to support them, and remove them when they are no longer needed. In other words, it enables scaling compute (query servers) and storage (HDFS) separately.

Also, with JethroData client-side load balancing, the ODBC/JDBC driver automatically routes each subsequent query to a different JethroData server in a round-robin fashion, evenly spreading the workload from concurrent queries.

JethroData for Amazon S3

Amazon S3 is a popular, low-cost storage service on the Amazon Web Services (AWS) cloud platform. It is a JethroData supported storage solution, so if you work with Amazon S3 you don't need to have a Hadoop cluster (and HDFS) as storage for JethroData.

When using Amazon S3, JethroData stores all its files (such as indexes) in a user-specified bucket on Amazon S3. When users want to query that data, they start one or more instances running JethroData and can *immediately start querying the data*. When they are done, they can just terminate the JethroData instances and stop paying for them.

No data copy phase

When starting a JethroData AWS instance, there is no need to copy all of the JethroData files from Amazon S3 to that instance or perform additional preparation. As the JethroData index is already stored on Amazon S3, a newly-started JethroData instance can immediately start serving queries. Let's quickly contrast it with two other, common analytics solutions on AWS.

➔ Amazon Redshift

Amazon Redshift is an AWS data warehouse service, based on a parallel database implementation optimized for full scans. If your data is stored on Amazon S3, you need to load it to Amazon Redshift each time you start a Redshift cluster. It may not be a problem if your Redshift cluster is up and running 24/7. However, if you want to run some queries on an existing, large data set, this massive copy and additional load step is costly in terms of both time and money.
In addition, while Redshift supports resizing your cluster, that procedure is not designed to be invoked frequently, as it generates a massive data copy that can take several hours and puts the cluster in a read-only mode. In contrast, JethroData servers are stateless and compute-only (Amazon S3 is used for storage) – so adding or removing servers is nearly instantaneous.

➔ Amazon Elastic MapReduce (temporary Hadoop cluster)

Some users deploy a temporary Hadoop cluster for queries, either taking advantage of the Amazon Elastic MapReduce (Amazon EMR) service or deploying their own. However, as with Amazon Redshift, this type of deployment requires a data copy step from Amazon S3 (where data is permanently stored) to the local Hadoop file system (HDFS). Again, that is costly in terms of both time and money.

Access Over Network

Amazon S3 is *always accessed over the network*. When you read or write files to Amazon S3, the data is moved between your instance and Amazon S3 over the network. In other words, with Amazon S3 there is no data locality – as compute and storage are always separated, you move the data to your code, by design. As such, it naturally fits the JethroData architecture and core philosophy of surgically fetching only the relevant data to a query host over the network.

Part 3: JethroData Performance Features

I/O-Related Features

JethroData I/O is performed over the network, whether to HDFS or to Amazon S3. As such, there are numerous I/O optimizations to minimize the required round-trips and bandwidth to the storage. The main ones are:

➔ Compressed columnar format

When data is loaded into the JethroData indexed format, it is broken into columns. Each column is stored separately, in a different set of files. At query time, only columns that participate in the query need to be accessed.
In addition, each column is encoded and compressed, minimizing its storage footprint.

➔ Efficient skip-scan I/O

Assume that a query needs to fetch a million values from several columns, after the `WHERE` clauses narrowed it down from billions of rows. The JethroData engine will skip-scan each column – for example, read a megabyte from the beginning, then skip ahead over a 12MB region, read another 16KB, and so on. Generally, it skip-scans each column from the beginning to the end, merging read requests if they are close by in the same file. This method automatically balances I/O size between network bandwidth consumption and network round-trips.
Alternatively, JethroData occasionally chooses to fetch an entire column (at a table or partition level) instead of using an index – for example, in case of a range scan across many distinct values. In those cases, JethroData uses block skipping whenever possible – analyzing the min/max values of each column block and skipping an entire block if it will not have any relevant rows. This is useful if the data is

physically clustered or sorted by that specific column.
Finally, JethroData always issues multi-threaded (parallel) I/O requests to efficiently utilize the bandwidth to the storage tier and to utilize multiple disks in parallel.

➔ **Local caching**

While JethroData issues efficient remote I/Os, it is still better not to repeatedly issue I/O requests to the same few popular columns, indexes and metadata files. For that reason, JethroData automatically caches locally frequently accessed column and index blocks, and also small metadata files. This reduces the number of round-trips to the HDFS NameNode and improves query performance with both Hadoop and Amazon S3.

Query Planning and Execution-Related Features

The JethroData engine includes many query planning and query execution optimizations. Some of them are based on leveraging indexes and some are based on understanding how BI tools generate SQLs on behalf of their users. Highlights include:

➔ **Smart optimizer choices based on indexing**

While many databases implement some sort of a cost-based optimization, they have a common challenge of reliance on collected statistics. Their optimizer has to take into account that the statistics it relies on might be inaccurate, stale or even missing, which complicates its optimizations. It also requires additional administrative steps of managing and tuning the statistics collection process and parameters.

In contrast, JethroData uses the index metadata instead of collected statistics for optimization. It is guaranteed to be always up-to-date, simplifying the optimization logic, avoiding optimization mistakes due to stale statistics and removing the need to manage statistics.

In addition, the JethroData engine uses index processing to make better decisions. For example, when joining tables, it is in many cases useful to change the join order so the biggest result set will be placed in the left side of a join. Using its indexes, the JethroData engine can in most cases know the exact size of the pre-join result set per table, even after multiple `WHERE` conditions, and use that information when deciding on join re-ordering.

➔ Answering queries from index metadata:

Many simple, common queries can be easily answered directly by processing the indexes, without the need to access the table's data. Some examples:

Getting a list of values for a column (common with BI tools – generating a drop down list):

```
SELECT distinct col FROM table;
```

Or alternatively, counting the number of rows per value (common for data exploration):

```
SELECT col, count(*) FROM table GROUP BY col
[HAVING...];
```

Counting the number of rows that match a complex condition:

```
SELECT count(*) FROM table
WHERE a=1 AND (b=2 OR d > 10) AND c IN (1,2,3);
```

➔ Parallel, pipelined execution

The JethroData execution engine is highly parallelized. The execution plan is made of many fine-grained operators, and the engine parallelizes the work within and across operators. In addition, the rows are pipelined between operators, and as the execution engine is multi-threaded, pipelining does not require memory copy, just passing a pointer.

➔ Adaptive query and index cache

When working with a BI tool, it is very common to have many repeating or mostly similar queries generated by dashboards and pre-built reports. Also, many ad-hoc explorations, while eventually diving deep into a unique subset of the data, typically start by progressively adding filters on top of the logical data model. The JethroData engine takes advantage of this predictable BI user behavior pattern by automatically caching result sets – both intermediate and near final ones – and transparently using them when applicable.

The adaptive cache is also incremental. For example, assume that query results were cached and later new rows are inserted to the fact table used by the query. If the query will be sent again, in most cases, JethroData will still leverage the cached results. It will compute the query results only on the delta and will merge it with the cached results to find the final query output. It will then replace the old cached results with the new ones. This mechanism is transparent to the user, who will never see stale or inconsistent data.

➔ Specific optimizations for BI-generated queries

The JethroData engine includes many optimizations that originated from analyzing common patterns in SQL statements generated by popular BI tools. These range from a simple, single index execution path (for example, getting a distinct list of column values for a dropdown menu) to more complex optimizations, such as star transformations and join eliminations.

Summary

We reviewed how JethroData's indexes are able to efficiently handle big data by leveraging an append-only design. We then showed how the JethroData architecture, which separates compute and storage, works well with both Hadoop and Amazon S3 storage. Finally, we covered some of JethroData's optimizations that enable interactive BI response times for business users.

.

More Information

For the latest information about our product, please visit:

http://www.jethrodata.com

To download the product for a free trial, please visit:

http://www.jethrodata.com/download

➔ Contact Us

Email: info@jethrodata.com

Twitter: http://twitter.com/jethrodata

About Jethro

Jethro was founded by a team of industry veterans committed to making big data analytics work in real time. Our passion is solving big problems, in this case building the technology that lets non-technical users interactively explore data on Hadoop and get immediate answers, using standard SQL or common BI tools.

In June 2015, Jethro closed an $8.1 million Series B financing round led by Square Peg Capital and existing investor Pitango Venture Capital.

Jethro is headquartered in New York City with an R&D office in Israel.