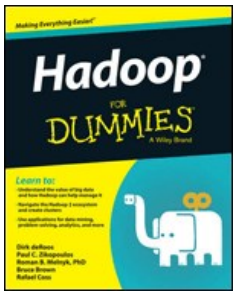


Hadoop for Dummies

by Dirk deRoos et al.

John Wiley & Sons (US). (c) 2014. Copying Prohibited.

Reprinted for Venkata Kiran Polineni, Verizon

kiran.polineni21@gmail.com

Reprinted with permission as a subscription benefit of **Skillport**,
http://skillport.books24x7.com/

All rights reserved. Reproduction and/or distribution in whole or in part in electronic,paper or other forms without written permission is prohibited.

Chapter 15: The Holy Grail—Native SQL Access to Hadoop Data

In This Chapter

- Seeing why SQL is important for Hadoop
- Looking at SQL access and the open source Hadoop community
- Evaluating proprietary SQL solutions

The NoSQL movement that has been happening over the past few years has taught two important lessons: a) Alternatives to relational databases can be a great help in solving a variety of problems and b) SQL isn't going anywhere. In fact, the NoSQL movement is now being rebranded as *NewSQL*, as in, "Here's a new technology where you can use SQL!" Even though we've seen a tremendous amount of innovation in the information management field — technologies are now available that can store graphs, documents, and key/value pairs at a massive scale — the IT market is still demanding SQL support for all of it. Hadoop is no exception, and a number of companies are investing heavily to drive open source projects and proprietary solutions for SQL access to Hadoop data.

SQL's Importance for Hadoop

There are compelling reasons that SQL has proven to be resilient. The IT industry has had 40 years of experience with SQL, since it was first developed by IBM in the early 1970s. With the increase in the adoption of relational databases in the 1980s, SQL has since become a standard skill for most IT professionals. You can easily see why SQL has been so successful: It's relatively easy to learn, and SQL queries are quite readable. This ease can be traced back to a core design point in SQL — the fact that it's a *declarative* language, as opposed to an *imperative* language. For a language to be declarative means that your queries deal only with the nature of the data being requested — ideally, there should be nothing in your query that determines *how* the processing should be executed. In other words, all you indicate in SQL is what information you want back from the system — not how to get it. In contrast, with an imperative language (C, for example, or Java, or Python) your code consists of instructions where you define the actions you need the system to execute.

In addition to the (easily leveraged) skills of your SQL-friendly IT professionals, decades' worth of database applications have also been built with SQL interfaces. As we discuss in Chapter 11, when talking about how Hadoop can complement the data warehouse, it's clear that organizations will store structured data in Hadoop. And as a result, they'll run some of their existing application logic against Hadoop. No one wants to pay for applications to be rewritten, so a SQL interface is highly desirable.

With the development of SQL interfaces to Hadoop data, an interesting trend is that commercial business analytics and data management tools are almost all jumping on the Hadoop bandwagon, including business intelligence reporting; statistical packages; Extract, Transform, and Load frameworks (ETL); and a variety of other tools. In most cases, the interface to the Hadoop data is Hive (see Chapter 13) or one of the other solutions described in this chapter.

Looking at What SQL Access Actually Means

When we use the term *SQL access*, we do so knowing that we're relying on a few basic assumptions:

- **Language standards**: The most important standard, of course, entails the language itself. Many "SQL-like" solutions exist, though they usually don't measure up in certain fundamental ways — ways that would prevent even typical SQL statements from working. The American National Standards Institute (ANSI) established SQL as an official technical standard, and the IT industry accepts the ANSI SQL-92 standard as representing the benchmark for basic SQL compliance. ANSI has released a number of progressively more advanced versions over the years as database technologies have evolved.
- **Drivers**: Another key component in a SQL access solution is the *driver* — the interface for applications to connect and exchange data with the data store. Without a driver, there's no SQL interface for any client applications or tools to connect to for the submission of SQL queries. As such, any SQL on Hadoop solution has to have JDBC and ODBC drivers at the very least, because they're the most commonly used database interface technologies.
- **Real-time access**: Until Hadoop 2, MapReduce-based execution was the only available option for analytics against data stored in Hadoop. For relatively simple queries involving a full scan of data in a table, Hadoop was quite fast as compared to a traditional relational database. Keep in mind that this is a batch analysis use case, where *fast* can mean hours, depending on how much data is involved. But when it came to more complex queries, involving subsets of data, Hadoop did not do well. MapReduce is a batch processing framework, so achieving high performance for real-time queries before Hadoop 2 was architecturally impossible. One early motivator for YARN, the new resource management and scheduling system on the block, was this need to support other processing frameworks to enable real-time workloads, such as interactive SQL queries. Indeed, a proper SQL solution should not leave people waiting for reasonable queries. (For more on YARN, see Chapter 7.)
- **Mutable data**: A common question in many discussions around SQL support on Hadoop is "Can we use `INSERT`, `UPDATE`, and `DELETE` statements, as we would be able to do in a typical relational database?" For now, the answer is no, which reflects the nature of HDFS — it's focused on large, immutable files. At the time of this writing, technologies such as Hive offer read-only access to these files. Regardless, work is ongoing in the Hive Apache project to enable `INSERT`, `UPDATE`, and `DELETE` statements.

SQL Access and Apache Hive

Reprinted for MC2WS/venkata.polineni@verizon.com, Verizon John Wiley & Sons (US), John Wiley & Sons, Inc. (c) 2014, Copying Prohibited

At the time of this writing, Apache Hive is indisputably the most widespread data query interface in the Hadoop community. (We cover Hive in depth in Chapter 13, describing its structure and how to use it.)

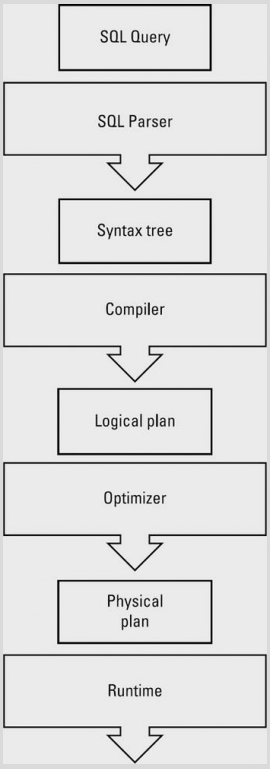
Originally, the design goals for Hive were not for full SQL compatibility and high performance, but were to provide an easy, somewhat familiar interface for developers needing to issue batch queries against Hadoop. This rather piecemeal approach no longer works, so the demand grows for real SQL support and good performance. Hortonworks responded to this demand by creating the Stinger project, where it invested its developer resources in improving Hive to be faster, to scale at a petabyte level, and to be more compliant to SQL standards. This work was to be delivered in three phases.

In Phases 1 and 2, you saw a number of optimizations for how queries were processed as well as added support for traditional SQL data types; the addition of the ORCFile format for more efficient processing and storage; and integration with YARN for better performance. In Phase 3, the truly significant evolutions take place, which decouple Hive from MapReduce. Specifically, it involves the release of Apache Tez (described in Chapter 7), which is an alternative processing model for Hadoop, designed for interactive workloads.

Massively parallel processing databases

To provide a better understanding of the SQL-on-Hadoop alternatives to Hive in this chapter, we thought it would be helpful to provide a primer on massively parallel processing (MPP) databases first.

As we explain in Chapter 13, Apache Hive is layered on top of the Hadoop Distributed File System (HDFS) and the MapReduce system and presents an SQL-like programming interface to your data (HiveQL, to be precise). This combination of Hadoop technologies deployed on a cluster is similar to MPP databases that have existed for a while in the IT marketplace. MPP databases usually provide an SQL interface and a relational database management system (RDBMS) running on a cluster of servers networked together by a high-speed interconnect. The figure shows the components of an RDBMS that are typically included in the SQL-on-Hadoop solutions described in this chapter.



Relational data systems have evolved considerably to a point where best practices have emerged among most offerings in terms of an optimal query execution infrastructure. The figure above shows this in terms of the flow of a query as it's processed by an RDBMS engine. First, the query text is parsed and understood. Then the syntax tree for the query is compiled into a logical execution plan, which is then optimized to form the final physical execution plan, which is then executed by the runtime. For many of the SQL-on-Hadoop solutions, we're seeing similar components being deployed in Hadoop.

MPP clusters are usually referred to as having a Shared-Nothing architecture, because each system has its own CPU, memory and disk. However, through the database software and high-speed interconnects, the system functions as a whole and can scale as new servers are added to the cluster. The overall system is explicitly tuned to provide fast, interactive query response. MPP databases are often more flexible, scalable, and cost effective than the traditional RDBMS, hosted on a large multiprocessor server.

In addition to the Stinger project, Hortonworks is spearheading an ambitious initiative to enable Hive to support editing data at the row level — in other words, enabling `INSERT`, `UPDATE`, and `DELETE` statements against Hive data with full compliance with the ACID properties for

Reprinted for MC2WS/venkata.polineni@verizon.com, Verizon John Wiley & Sons (US), John Wiley & Sons, Inc. (c) 2014, Copying Prohibited

database systems: Atomicity, Consistency, Isolation levels, and Durability. (For more on the ACID properties, see Chapter 11.)

Solutions Inspired by Google Dremel

For most people, the term *Dremel* brings to mind a handy high-speed, low-torque tool that works well for a variety of jobs around the house. But did you know that Google created a Dremel? Rather than produce *another* handheld mechanical tool, though, Google chose a fast software tool intended for interactive analysis of big data. As with other Google technologies that inspired parts of the Hadoop ecosystem, such as MapReduce (see Chapter 6), Google File System (HDFS, see Chapter 4), and BigTable (see HBase, Chapter 12), Google developed Dremel for use internally and then published a paper describing the purpose and design of the technology. (In other words, Dremel is not something you can download and use on your Hadoop cluster.)

TIP You can find Google's Dremel whitepaper at this site:

http://research.google.com/pubs/pub36632.html

Google uses Dremel for a variety of jobs, including analyzing web-crawled documents, detecting e-mail spam, working through application crash reports, and more. Google's BigQuery service actually uses Dremel.

As we discuss in Chapter 1, Google designed MapReduce technology for batch processing over massive sets of data. As their needs evolved, so did their technology, and Google decided to create Dremel to improve performance for interactive queries against big data sets. The MapReduce approach provides scalability and query fault tolerance, but it's fundamentally a batch-based system, so response times for smaller queries (queries involving only a small part of an entire data set, for instance) are often not what users expect. So Google developed a query execution technology designed for interactive queries, which runs on intermediate servers on top of the Google File System (GFS). (Remember, GFS was the inspiration for Apache HDFS, which is Hadoop's file system.)

Similar to Hive, Dremel uses an SQL-like language (familiar to most programmers) and employs a columnar data layout. Dremel provides fast, interactive query response while preserving the scalability and fault tolerance found in Apache Hive. In the Dremel whitepaper, Google explains how it can perform aggregation queries within seconds over tables with a trillion rows — not bad at all.

So Google has its Dremel technology, which it uses internally, but then there are all the technologies "inspired by" Dremel (kind of like all those perfumes "inspired by" Drakkar Noir). We introduce you now to two "inspired by" products — Apache Drill and Cloudera Impala. The pattern here is similar in both cases:

Apache Drill

As of this writing, Drill is a candidate project in the Apache incubator. We don't mean that Apache Drill is especially sickly, though. The Apache Software Foundation (ASF) candidate technologies all begin as incubator projects before becoming official ASF technologies. You can read about the Apache Incubator at

http://incubator.apache.org

You can read about Drill at

http://incubator.apache.org/drill

Inspired by Google's Dremel technology, the stated performance goal for Drill is to enable SQL queries against a petabyte or more of data distributed across 10,000-plus servers. Figure 15-1 illustrates the architecture of Apache Drill.

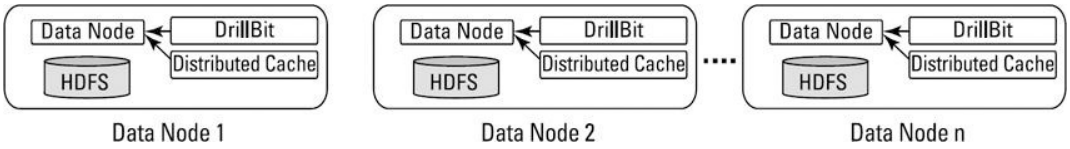


Figure 15-1: Apache Drill architecture

In Figure 15-1, we see that the key to the Drill architecture are the DrillBit servers deployed on each data node. Note that each server includes a query parser, compiler, optimizer, and runtime, but there is a master DrillBit server nominated by Zookeeper servers, which oversees the execution of the queries and looks after the task of pulling together the interim result sets into a single set of output.

Like Dremel, Drill can coexist with, and complement, MapReduce, but MapReduce isn't used to fulfill queries, as with Apache Hive. Instead, execution engines called *Drillbits* have been developed by members of the Drill community. This community aims to provide low-latency queries for applications such as real-time business intelligence dashboards, fraud detection, and other time-sensitive use cases. Drill supports nested data types such as Avro, JSON, and Google protocol buffers. These nested data types allow for very large denormalized tables. The Drill development team is also working on providing extensive SQL support by targeting SQL2003 compliance. Finally, note that the Drill team is providing HBase support so that users will be able to query HBase tables with SQL.

Cloudera Impala

Cloudera is a leading Apache Hadoop software and services provider in the big data market. Like Apache Drill, Cloudera's Impala technology seeks to improve interactive query response time for Hadoop users. As we discuss in Chapter 13, Apache Hive has provided a familiar and

Reprinted for MC2WS/venkata.polineni@verizon.com, Verizon John Wiley & Sons (US), John Wiley & Sons, Inc. (c) 2014, Copying Prohibited

powerful query mechanism for Hadoop users, but query response times are often unacceptable due to Hive's reliance on MapReduce. Cloudera's answer to this problem is Impala. Cloudera has developed an MPP query engine, written in C++, to replace the MapReduce layer leveraged by Apache Hive. Unlike Dremel and Drill, Cloudera decided that a native C++ MPP engine — instead of a Java engine — was the answer for fast, interactive Hadoop queries.

Note that Impala uses HiveQL as a programming interface, and Impala's Query Exec Engines are co-located with HDFS data nodes, in keeping with the Hadoop approach of co-locating data with processing tasks. Impala can also use HBase as a data store. In this sense, Impala is an extension to Apache Hadoop, providing a very high-performance alternative to the Hive-on-top-of-MapReduce model.

TECHNICAL STUFF In Chapter 13, we present several Hive file formats: `TEXTFILE`, `SEQUENCEFILE`, `RCFILE`, and `ORC`. Cloudera and Twitter led the development of the new Hadoop file format `PARQUET`, which can be used with Impala and is available as open source on GitHub. The Parquet file format provides a robust columnar medium for storing data in Hadoop. It supports highly efficient compression and encoding, and is effective for storing nested data structures.

You can find Cloudera's Impala technology, which also was inspired by Google's Dremel invention, at https://github.com/cloudera/impala.

IBM Big SQL

IBM has a long history of working with SQL and database technology, as the introduction to this chapter makes clear. In keeping with this history, IBM's solution for SQL on Hadoop leverages components from its relational database technologies that are ported to run on Hadoop.

If you're at all familiar with IBM's product naming for its Big Data products and features, you can easily guess what they've named their SQL on Hadoop solution: Big SQL. The goal of Big SQL is to provide a SQL interface on Hadoop that gives users as much as possible of what they're used to with SQL interfaces for relational databases. This means extensive query syntax support, fast performance that doesn't require users having to monkey with their queries, and the ability to control data security.

Figure 15-2 shows a partial deployment of BigInsights, IBM's Hadoop distribution running Big SQL.

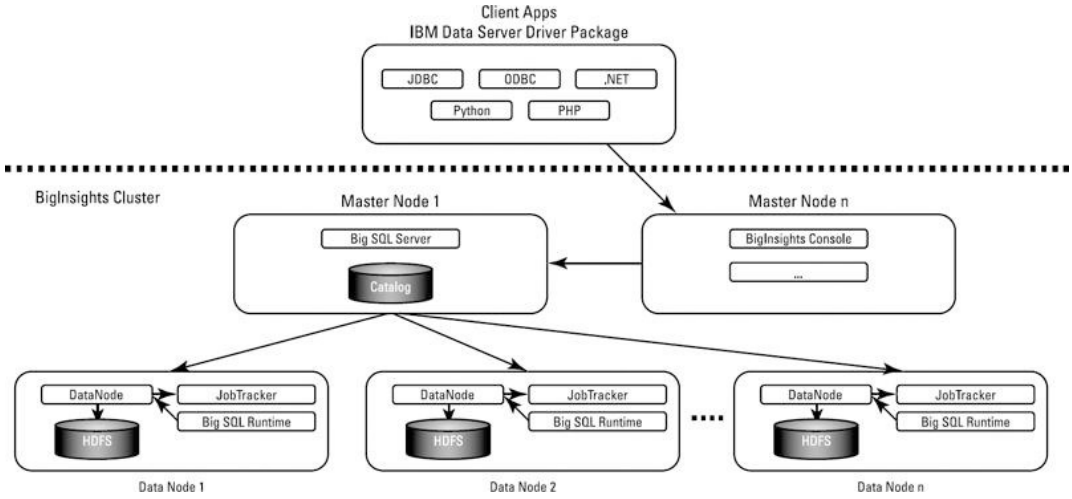


Figure 15-2: IBM Big SQL architecture

In Figure 15-2 you can see a subset of the master nodes and data nodes behind the BigInsights firewall. One of the master nodes is running the Big SQL server, which includes IBM's SQL compiler and optimizer. Also included on this master node is a catalog, where metadata and statistics about any cataloged data in HDFS is stored for use by the compiler/optimizer. Subsections of queries are sent to the applicable data nodes where requested data is stored, and there the Big SQL Runtime (which is IBM's SQL runtime) executes the workload. Rather than run mapper and reducer processes and persist files with intermediate result sets, Big SQL uses continuously running daemons that pass messages between each other. It's important to note that the data being queried is stored and managed by Hadoop. Big SQL supports standard Hadoop file formats — for example, RCFile and Parquet.

Big SQL provides the same extensive SQL support as the IBM relational database products — for example, ANSI SQL-2011, and compatibility for IBM's SQL Procedural Language (SQL/PL). (At the time of writing, IBM was working on providing support for Oracle's SQL dialect and their PL/SQL procedural language.) Along with the standard IBM SQL engine come a number of other capabilities, most notably IBM's row- and column-based security (also known as Fine-Grained Access Control, or FGAC), where only specific users can be authorized to see certain sets of data rows or columns.

Big SQL comes with the standard IBM Data Server Client, which includes a driver package (refer to Figure 15-2). Traditional database applications can connect to the BigInsights Hadoop cluster and securely exchange encrypted data over SSL.

Pivotal HAWQ

In 2010, EMC and VMware, market leaders in delivering IT as a service via cloud computing, acquired Greenplum Corporation, the folks who had successfully brought the Greenplum MPP Data Warehouse (DW) product to market. Later in 2012, Pivotal Labs, a leading provider of Agile software development services, was also acquired. Through this federation of companies, the Pivotal HD Enterprise platform was

Reprinted for MC2WS/venkata.polineni@verizon.com, Verizon John Wiley & Sons (US), John Wiley & Sons, Inc. (c) 2014, Copying Prohibited

announced in early 2013. This platform, which is integrated with Apache Hadoop, includes the Pivotal HAWQ (Hadoop With Query) product — the former Greenplum MPP DW product. Though the Pivotal HD Enterprise platform also includes other components and technologies (VMware's GemFire, for example), we want to draw your attention to the Pivotal HAWQ product, Pivotal's approach to low-latency interactive SQL queries on Hadoop. Pivotal has integrated the Greenplum MPP Shared-Nothing DW with Apache Hadoop to enable big data analytics. The Pivotal HAWQ MPP DW stores its data in the Apache HDFS.

Pivotal HAWQ provides ANSI SQL support and enables SQL queries of HBase tables. HAWQ also includes its own set of catalog services instead of using the Hive metastore. The Pivotal HAWQ approach is to provide a highly optimized and fast Hadoop SQL query mechanism on top of Apache Hadoop.

Hadapt

Late in the year 2010, Hadapt was formed as a start-up by two Yale University students and an assistant professor of computer science. Professor Daniel Abadi and Kamil Bajda-Pawlikowski, a PhD student from Yale's computer science department, had been working on the research project HadoopDB. After this paper was published, Justin Borgman, a student from the Yale School of Management, became interested in the work. He would later team up with Professor Abadi and Kamil Bajda-Pawlikowski to form Hadapt.

The Hadapt strategy is to join Apache Hadoop with a Shared-Nothing MPP database to create an adaptive analytics platform. This approach provides a standard SQL interface on Hadoop and enables analytics across unstructured, semistructured, and structured data on the same cluster.

Like Apache Hive and other technologies, Hadapt provides a familiar JDBC/ODBC interface for submitting SQL or MapReduce jobs to the cluster. Hadapt provides a cost-based query optimizer, which can decide between a combination of MapReduce jobs and MPP database jobs to fulfill a query, or the job can be handled by the MPP database for fast interactive response. By joining an Apache Hadoop cluster with an MPP database cluster to create a hybrid system, Hadapt solves the query response time and partial SQL support (via HiveQL) found in Apache Hive.

The SQL Access Big Picture

SQL access to Hadoop data is a burning (and ongoing) concern. Many vendors are offering solutions — some are adding value to the Hadoop ecosystem by writing their own high-performance MPP engine to replace the higher-latency MapReduce system, and others are working hard to improve the performance of MapReduce by rewriting parts of the Hadoop system with native code (using the C and or C++ languages, for example) instead of with Java. Some have decided that integrating Shared-Nothing MPP database systems and Hadoop on the same platform is the way to go. History has shown that in technological battles such as this one, only one or two victors will emerge, leaving many solutions obsolete. The positive perspective in this case is that regardless of the specific winning technology, the interface will at least be SQL.

Reprinted for MC2WS/venkata.polineni@verizon.com, Verizon John Wiley & Sons (US), John Wiley & Sons, Inc. (c) 2014, Copying Prohibited