Apache Hive: A SQL Engine for the Apache Hadoop Framework, and More

Many people who are familiar with the Apache Hadoop framework think of Hive as a SQL engine—a way to automatically compile a SQL query into a set of MapReduce jobs and then run them on a Hadoop cluster. While this is accurate, the things that make Hive really novel are its facilities for data and metadata management.

MapReduce is a very flexible programming paradigm, but most users find it too low-level for everyday data analysis tasks. Almost from the day Hadoop was introduced, people began looking for ways to express their data analysis tasks using higher-level abstractions built on top of MapReduce. The engineers at Facebook who built the first version of Hive decided to use SQL as their higher-level language due to its widespread adoption, and also because a majority of their analysts already knew how to use it.

Anyone with even a little bit of prior SQL experience will be able to come up to speed quickly with Hive. Hive supports large portions of the SQL-92 standard, as well as several extensions that are designed to make it easier to interact with the underlying Hadoop platform.

SQL databases organize data into tables, each of which is split into typed columns. Hive is no different, but unlike a traditional database, Hive does not require you to define the structure of the table before importing data. Instead, Hive allows you to project tabular structure onto the underlying data at query execution time—a feature that’s frequently referred to as “schema-on-read.” The two components that make schema-on-read possible are the Hive SerDe layer and the Hive MetaStore.

• **Hive SerDes.** Experienced database users are often surprised to hear that there’s no “Hive” storage format. In fact, Hive was designed to be completely format agnostic, and accomplishes this feat by supporting pluggable storage format handlers called SerDes. SerDe stands for Serialize/Deserialize, and that’s basically what they do. When operating in serialization mode, a SerDe converts the rows in a table to a specific physical storage format; for example, JavaScript Object Notation (JSON), comma-separated value (CSV), or Apache Avro formats. In deserialization mode, a SerDe performs the same steps in reverse, extracting tabular records from an underlying file. Hive ships with a standard SerDe library that provides support for many of the most popular Hadoop formats including CSV, JSON, Avro, SequenceFile, and RCFile formats. Also, since SerDes are public application programming interfaces (APIs), you have the option of writing your own SerDe if you need to support a new file format.

Carl Steinbach, committer and Project Management Committee member on the Apache Hive project and a software engineer from Citus Data, talks about Apache Hive.

Apache Hive* Community Spotlight

Apache Hive*

“One of the major value propositions for Apache Hive is that it can act like an adapter between the Hadoop platform and the large ecosystem of data analysis tools that are designed to run on top of relational databases.”

—Carl Steinbach
Hive MetaStore. The MetaStore serves as Hive’s table catalog. In conjunction with SerDes, the MetaStore allows users to cleanly decouple their logical queries from the underlying physical details of the data they are processing. The MetaStore accomplishes this by maintaining lookup tables for several different types of catalog metadata.

- **Table-to-columns:** This mapping defines the logical structure of a Hive table consisting of column names and column types.
- **Table-to-storage-location:** This mapping allows Hive to convert a table name to a set of storage locations on an Apache* HDFS* file system. For example, a user could create an "employees" table that points to data stored in a collection of CSV files located in the HDFS directory "/hive/warehouse/employees."
- **Table-to-SerDe:** This mapping allows Hive to determine which SerDe it should use when reading and writing data to a particular table. Since SerDes are tied to the underlying data format, this is equivalent to defining the data format of the table.

Since MetaStore persists these mappings, you can use logical table names like "employees" instead of constantly repeating the definition of the table at the beginning of every SQL script. This saves you time, makes it easier for you to share your data with colleagues, and results in more maintainable scripts, since you can decouple your analysis logic from the underlying physical details of the data.

Hive and Existing Warehouse Infrastructure

Hive has developed to the point where, with some effort, you can use it to replace existing data warehouse infrastructure—and one of the long-term goals of the project is for people to be able to replace large portions of their data warehouse infrastructure with Hive.

One of the major value propositions for Hive is that it can act like an adapter between the Hadoop platform and the large ecosystem of data analysis tools that are designed to run on top of relational databases. These tools include extract, transform, and load (ETL) applications; business intelligence (BI) apps; and SQL integrated development environments (SQL IDEs). Since it speaks SQL and supports Open Database Connectivity (ODBC) and Java* Database Connectivity (JDBC) drivers, Hive makes it possible to run many of these same tools on top of the Hadoop framework. This makes it much easier for organizations to adopt Hadoop because it eliminates the cost of retraining employees on new tools.

Hive Limitations

Hive has inherited certain limitations from HDFS and MapReduce. First, because HDFS is a write-once, read-many-times file system, Hive can’t perform row-level inserts, updates, or deletes. As a result, Hive cannot be used for online transaction processing (OLTP) and is more suited to online analytical processing (OLAP) workloads.

Second, Hive queries have a minimum latency built into every job. When Hive compiles a query into a set of MapReduce jobs, it actually has to launch and coordinate those jobs on the cluster. This means that even small queries will take upwards of ten seconds to finish. On the other hand, because it scales so easily, you can run queries on a terabyte of data fairly quickly by distributing the load over all the nodes in your cluster. Because of these two characteristics, some people describe Hive as a high-latency, high-throughput system.

Is Apache Hive* Compliant with the SQL Standard?

The community is working hard to bring Apache Hive* in line with the American National Standards Institute (ANSI) SQL-92 standard. A number of different people are contributing resources to make this happen, but so far it’s still a work in progress. We are also working to add support for some of the new features that were introduced in more recent SQL standards, including analytic SQL functions.

Hive* will probably never support OLTP-type SQL, in which the system updates or modifies a single row at a time, due to limitations of the underlying Apache* Hadoop* Distributed File System.

Note: ANSI SQL-92 is the third revision of the SQL database query language.

Apache Pig* versus Hive

Up until recently, the single biggest difference between Apache Pig* and Hive was that Pig* did not support persistent metadata, and consequently forced you to define your data sources at the beginning of every PigLatin script. The Apache HCatalog project was started with the goal of making Hive’s metadata accessible to other Hadoop ecosystem projects by providing a set of wrapper APIs around Hive’s MetaStore and SerDe layers. Current development for Pig includes adding support for metadata. Besides making PigLatin more convenient to use, this will help to facilitate sharing data between Pig and Hive, since users of both systems will see the same tables.
When would you use Hive over Pig? It's really just a matter of which language you feel more comfortable using. Developers who already use the Java language or a scripting language like Perl will probably find the PigLatin imperative style more familiar, while Hive is an obvious choice for people with prior SQL experience. While some claim that PigLatin is a better fit for ETL tasks, the fact that several major ETL tool vendors have built their Hadoop integrations on top of Hive would seem to contradict this statement. One advantage of Hive, however, is that at the moment it's the only component in the Hadoop ecosystem that provides JDBC and ODBC drivers, which in turn allow you to use Hive with a large collection of existing data analysis tools.

**Next Steps for Hive**

Work on Hive is moving rapidly. Long-term, I think people will begin to view their Hadoop cluster as their data warehouse. Several things are driving this, including:

- **Hadoop is a lot more cost-effective.** It runs on commodity hardware, it's open source, it’s free, and organizations aren’t locked into a certain vendor’s software or hardware. This also encourages people to retain more data than they would otherwise.

- **Hadoop scales out very easily.** If you’re not meeting service level agreements (SLAs) for a certain job, you can increase the size of your cluster rather than buy more expensive proprietary hardware.

A second trend I’m seeing is that BI and ETL tool vendors will start to abandon the building of their own proprietary cluster infrastructure and use MapReduce and HDFS instead. In fact, we are already starting to see this with some vendors who are building solutions on top of Hive and Pig.

This paper is derived from an interview with Carl Steinbach on August 22, 2012. For the full interview, [listen to the podcast](http://hive.apache.org/).