HadoopDB

a major step towards a dead end
Thema 5
Seminar 01912
Sommersemester 2011

Referent: Thomas Koch

June 23, 2011

Contents

This paper presents and discusses two texts about HadoopDB[ABPA+09] and the Apache Hive[TSJ+09] project which is used by the former.

1 HadoopDB

1.1 Scope and Motivation

The HadoopDB project aims to combine the scalability advantages of MapReduce with the performance and efficiency advantages of parallel databases. Parallel databases in this context are defined as "analytical DBMS systems [sic] that deploy on a shared-nothing architecture". Current parallel databases are usually scaled only into the tens of nodes. It may be impossible to scale them into hundreds or thousand nodes like MapReduce for at least three reasons:

1. Failures become frequent at this scale but the systems don't handle them well.
2. A homogeneous cluster of machines is assumed which is impossible to provide at higher scale.
3. The systems has not yet been tested at this scale.

MapReduce on the other hand is known to scale well into thousands of nodes. However, according to work done by Stonebraker and others, MapReduce would neither be suitable for analytical work[DD08] nor perform well[PPR+09].

A further advantage of HadoopDB should be low cost. This could be achieved by building on existing open source solutions: The MapReduce implementation Apache Hadoop, Apache Hive and the RDBMS PostgreSQL. It may however be argued whether these components could be "used without cost" as claimed. The production use of software like Hadoop and PostgreSQL still requires highly qualified and payed personal.

Desired Properties The following list presents the desired properties of the HadoopDB project together with first comments. It is argued that neither MapReduce nor parallel databases would provide all of these properties.

- Performance: It is claimed, that performance differences could "make a big difference in the amount, quality, and depth of analysis a system can do". Also performance could significantly reduce the cost of an analysis if fewer hardware ressources were used. However Stonebraker and Abadi concluded already in 2007 that the primary factor for costs has shifted from hardware to personal.[SMA+07, 2.5 No Knobs] It should also be considered that the costs for licenses of proprietary database systems often outrun hardware costs by orders of magnitude. Finally it is not at all certain, that in practice performance differences are a limitation factor for the range of possible computations.

  The paper suggests that current practice in analytical systems would be to load data in a specialized database and to make calculations while an operator is waiting for the results. In MapReduce systems however calculations are directly done on the original data and could therefor run continuously during the normal operation of the system. The performance of calculations may therefor not be a practical problem at all.

- Fault Tolerance: The probability, that at least one cluster node fails during a computation may be negligible for less then 100 nodes. With growing data size however the cluster needs to grow. Therefor the probability of a single node failure grows as well

as the duration of algorithms. This continues until it's not possible anymore to finish calculations without a single node failing.

A system that restarts the whole computation on a single node failure, as typical in RDBMSs, may never be able to complete on large enough data sizes.

- Ability to run in a heterogeneous environment: Most users of a large cluster would not be able to provide a large number of totally identical computer nodes and to keep there configuration in sync. But even then the performance of components would degrade at different pace and the cluster would become heterogeneous over time. In such an environment work must be distributed dynamically and take node capacity into account. Otherwise the performance could become limited by the slowest nodes.

- Flexible query interface: The system should support SQL to integrate with existing (business intelligence) tools. Ideally it also supports user defined functions (UDF) and manages their parallel execution.

The above list does not include energy efficiency as a desired property although it is one of the most important subjects in recent discussions about data centers.[1] It has also been argued, that standard performance benchmarks should be enhanced to measure the performance relative to energy consumption.[FHH09]

Section 4 describes the parallel databases and MapReduce in further detail. From the desired properties the former would score well on performance and the flexible interface and the later on scalability and the ability to run in a heterogeneous environment. A paper of Stonebraker, Abadi and others[PPR+09] is cited for the claim of MapReduces lack of performance. This paper is cited over twenty times in the full HadoopDB text and is also the source for the benchmarks later discussed. Other evidence for a lack of performance of MapReduce is not provided.

It is also recognized that MapReduce indeed has a flexible query interface in that its default usage requires the user to provide map and reduce functions. Furthermore the Hive project provides a SQL like interface to Hadoop. Thus although it isn't explicitly said it could be concluded that MapReduce also provides a flexible query interface.

It can be summed up at this point, that MapReduce is excellent in two of four desired properties, provides a third one and only lacks in some degree in performance. Hadoop is still a very young project and will certainly improve its performance in the future. It may be asked, whether a project like HadoopDB is still necessary, if Hadoop itself already provides nearly all desired properties.

It is interesting, how the great benefit of not requiring a schema up-front is turned into a disadvantage against MapReduce:[ABPA+09, 4.2]

> By not requiring the user to first model and load data before processing, many of the performance enhancing tools listed above that are used by database systems are not possible. Traditional business data analytical processing, that have standard reports and many repeated queries, is particularly, poorly suited for the one-time query processing model of MapReduce.

Not requiring to first model and load data provides a lot of flexibility and saves a lot of valuable engineering time. This benefit should be carefully evaluated and not just traded against performance. The quote also reveals another possible strategy to scale analytical processing: Instead of blaming MapReduce for its favor of one-time query processing, the analytical software could be rewritten to not repeat queries. It'll be explained later that Hive provides facilities to do exactly that by allowing the reuse of SELECT statements.

[1] see Googles Efficient Data Center Summits `http://www.google.com/corporate/datacenter/summit.html` (2011-06-19) and Facebooks open compute project `http://opencompute.org` (2011-06-19)

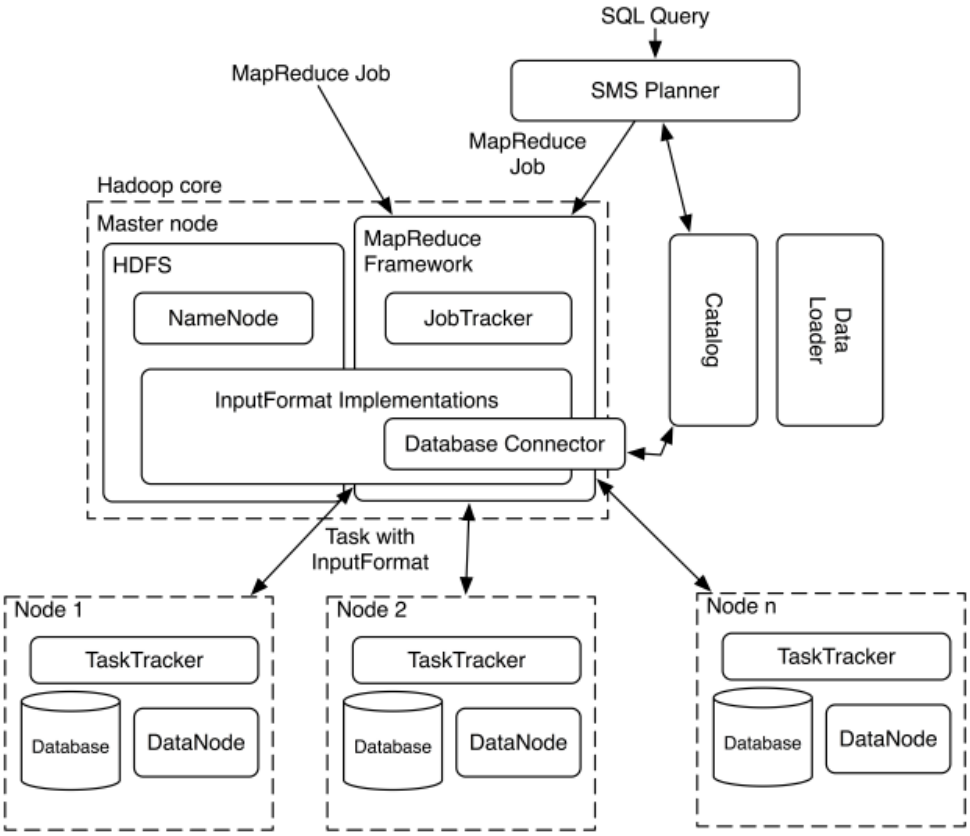

Figure 1: The Architecture of HadoopDB[ABPA+09]

1.2 Architecture

Hive is a tool that provides a SQL interface to MapReduce. It translates SQL queries into apropriate combinations of map and reduce jobs and runs those over tabular data stored in HDFS. HadoopDB extends Hive in that HDFS is replaced by node local relational databases.

Data Loader HadoopDB comes with two executable java classes and a python script[2] necessary to partition and load the data into HadoopDB. The data loading is a cumbersome process with several manual steps. Most of the steps must be executed in parallel on all nodes by some means of parallel ssh execution. This leaves a lot of room for failures. It is also necessary to specify the number of nodes in advance. A later change in the number of nodes requires to restart the data load process from scratch. It is also not possible to incrementally add data.

The following list describes the necessary steps to load data into HadoopDB. At each step it is indicated how often the full data set is read (r) and written (w). It is assumed, that the data already exists in tabular format in HDFS and that HDFS is used for this process with a replication factor of 1. This means that in case of a hard drive failure during the import process the process would need to be started again from the beginning. A higher replication factor would lead to even higher read and write numbers in the following list.

1. Global Hasher: Partition the data set into as many parts as there are nodes in the cluster. (2r, 2w + 1 network copy)[3]

[2] `http://hadoopdb.sourceforge.net/guide` (2011-06-20)
[3] The result of the map phase is written to disk and from there picked up by the reduce phase.

2. Export the data from HDFS into each nodes local file system. (1r, 1w)[4]

3. Local Hasher: Partition the data into smaller chunks. (1r, 1w)

4. Import into local databases. (1r, 1w + secondary indices writes)

We can see, that before any practical work has been done, HadoopDB implies five full table scans and writes. In the same time at least 2.5 map and 2.5 reduce jobs could have completed. If we assume that the results of typical map and reduce jobs are usually much smaller then the input, even more work could have been done. There are indications that it may be possible to optimize the data loading process to lower the number of necessary reads and writes. It will be discussed in the benchmark section, why this doesn't seem to be of high priority for the HadoopDB authors.

Neither the HadoopDB paper nor the guide on sourceforge make it totally clear what the purpose of the Local Hasher is. It is suggested that a partitioning in chunks of 1GB is necessary to account for practical limitations of PostgreSQL when dealing with larger files.[5]

SQL to MapReduce to SQL (SMS) Planner The heart of HadoopDB is the SMS planner. Hive translates SQL like queries into directed-acyclic graphs (DAGs) of "relational operators (such as filter, select (project), join, aggregation)". These operator DAGs (the query plan) are then translated to a collection of Map and Reduce jobs and sent to Hadoop for execution. In general before each JOIN or GROUP BY operation data must be shared between nodes.

The HadoopDB project hooks into Hive just before the operations DAG gets sent to Hadoop MapReduce.(Figure 1) The planner walks up the DAG and transforms "all operators until the first repartitioning operator with a partitioning key different from the database's key" back into SQL queries for the underlying relational database.

For those SQL queries where HadoopDB can push parts of the query into the database one could expect a performance gain from secondary indices and query optimization. For other SQL queries one should expect the same performance as with plain Hive. In contrast to Hive HadoopDB can and does take advantage of tables that are repartitioned by a join key and pushes joins over these keys in the database layer.

Auxiliary Components HadoopDB includes code to connect to the local databases and a catalog that stores database connection parameters and metadata "such as data sets contained in the cluster, replica locations, and data partitioning properties". The catalog must also be generated manually from two input files which adds to the work necessary to setup the data.

1.3 Benchmarks

Five different tasks are provided to benchmark two parallel databases (Vertica, DBMS-X), Hadoop and HadoopDB on three different number of nodes (10, 50, 100). The tasks are executed under the assumption of no node failures. Thus the replication factor of HDFS respectively the number of replicas in HadoopDB is set to 1 and the task run is not counted in case of a node failure. In a later task run node failures are also taken into account.

The tasks are borrowed from the already mentioned Stonebraker paper[PPR+09]. It is therefor necessary to refer to Stonebrakers work for their description at some points.

```
CREATE TABLE Documents (
  url VARCHAR(100) PRIMARY KEY,
  contents TEXT
);

CREATE TABLE Rankings (
  pageURL VARCHAR(100) PRIMARY KEY,
  pageRank INT,
  avgDuration INT
);
```

```
CREATE TABLE UserVisits (
  sourceIP VARCHAR(16),
  destURL VARCHAR(100),
  visitDate DATE,
  adRevenue FLOAT,
  userAgent VARCHAR(64),
  countryCode VARCHAR(3),
  languageCode VARCHAR(6),
  searchWord VARCHAR(32),
  duration INT
);
```

Figure 2: Tables used for the analytical benchmark tasks.

1.3.1 Data Sets

The first task (Grep) uses a simple schema of a 10 bytes key and a random 90 bytes value field. The next three analytical tasks work on a more elaborate schema of three tables. (Figure 2) This schema was originally developed by Stonebraker and others to argue that parallel databases could also be used in areas where MapReduce recently became popular.[PPR+09, 1.] The Documents table should resemble the data "a web crawler might find" while the UserVisits table should "model log files of HTTP server traffic". The Rankings table is not further described.[PPR+09, 4.3]

It seems that Stonebraker has a very naive idea of the inner workings of a typical web crawler and the information typically logged by a HTTP server. The BigTable paper from 2006 explains, that Google stores multiple versions of crawled data while a garbage collector automatically removes the oldest versions.[CDG+06] This requirement alone would be very complicate to model on top of a relational data store. Furthermore a web crawler would certainly store a lot more data, for example the HTTP headers sent and received, which may be several pairs due to HTTP redirects or a history when the page was last loaded successfully along with the last failed loading attempts and the reasons for those failures. Subsequental processes enrich the data for example with data extracted from HTML or the HTTP headers and with a list of web sites pointing to this site (inlinks) together with the anchor text and the page rank of the inlinks. The pageRank column from the Rankings table would surely be inlined into the Documents table.

The UserVisits table is likely unrealistic. A HTTP server usually has no knowledge of any adRevenue related to a web site just served, nor is anything known about a searchWord. Web sites that highlight search words previously entered in a search engine to find that page do so by parsing the HTTP referrer header. A HTTP server is usually not capable to extract this information. The same is valid for the HTTP accept language header which provides a languageCode. A countryCode can usually only be obtained by looking up the users IP address in a so called geoIP database.

The most important flaw in the data sets however is the assumption of a separated, off line data analytics setting. Todays search engines need to reflect changes in crawled pages as soon as possible. It is therefor typical to run the whole process of crawling, processing and indexing on the same database continuously and in parallel.[PD10]

It is understandable that Abadi choose to reuse the benchmarks to enable comparison. However he wants to demonstrate the applicability of HadoopDB "for Analytical Workloads".

[4] It is not clear, why this step is necessary. The reduce output from the previous step could have been written directly to the local file systems of the nodes running the reduce jobs.

[5] "We recommend chunks of 1GB, because typically they can be efficiently processed by a database server."

One should therefore expect benchmarks that use schemes related to financial, customer, production or marketing data.

Data Loading It has already been described how complicated the data loading procedure is. The benchmark measured a duration of over 8.3 hours for the whole process. Not accounted for is the manual work done by the operator during the individual steps to start each individual step, let alone the provisioning and preparation of virtual machines. Since the process is not (yet) automatized and may even need to be restarted from scratch in case of failures, it can in practice take two working days. The HadoopDB authors however do not consider this to be a problem[ABPA+09, 6.3]:

> While HadoopDB's load time is about 10 times longer than Hadoop's, this cost is amortized across the higher performance of all queries that process this data. For certain tasks, such as the Join task, the factor of 10 load cost is immediately translated into a factor of 10 performance benefit.

This statement however ignores that data is often already stored on Hadoop in the first place and that Hadoop wouldn't require a load phase at all. Instead Hadoop could run analytical tasks incrementally or nightly and provide updated results while HadoopDB still struggles to load the data.

1.3.2 Grep Task

The Grep Task executes a very simple statement over the already described key value data set:

```
SELECT * FROM Data WHERE field LIKE '%XYZ%';
```

Hadoop performs in the same area as HadoopDB and the parallel databases require less then half the time. It is argued that this difference in performance in such a simple query is mainly due to the use of data compression which results in faster scan times. Although Hadoop supports compression and could therefore share the same benefits, it isn't used. An explanation is found in the Stonebraker paper. They did some experiments with and without compression and in their experience Hadoop did not benefit from compression but even get slower.

However it seems that there could be mistakes in their usage of Hadoop. Instead of using the Hadoop SequenceFile format to write compressed data, they split the original files in smaller files and used a separate gzip tool to compress them. There is no explanation why they did the splitting nor do they provide the compression level used with gzip. It is very likely that the raise in the number of files caused by the splits led to a higher number of necessary disk seeks.

They also tried record-level compression. This is however pointless since the values in the Grep Task only have a length of 90 bytes and may even get larger from the compression header.

1.3.3 Selection Task

The selection task executes the following query:

```
SELECT pageUrl, pageRank FROM Rankings WHERE pageRank > 10;
```

The description and result diagram for this task is a bit confusing. The text says that HadoopDB would outperform Hadoop while the diagram on first sight reports comparable

execution times for both. The solution is that there is a second bar for HadoopDB reporting the duration for a data repartitioning that has been optimized for this task.

It is questionable whether the long and complicated data loading is still tolerable, if it even has to be optimized for different queries. One would expect that the upfront investment of data modeling and loading would be rewarded by a great variety of possible queries that could be executed afterward with high performance.

1.3.4 Aggregation Tasks

These two tasks differ only in their grouping on either the full sourceIP or only a prefix of it.

```
SELECT SUBSTR(sourceIP, 1, 7), SUM(adRevenue) FROM UserVisits
GROUP BY SUBSTR(sourceIP, 1, 7);

SELECT sourceIP, SUM(adRevenue) FROM UserVisits
GROUP BY sourceIP;
```

Hadoop and HadoopDB differ only in around 20% of their execution time while the parallel databases are again significantly faster. It is once again argued that this would be mainly caused by compression and therefor the same arguments apply as with the Grep Task.

Furthermore this tasks points out the advantage of a high level access language. Hive (and thereby HadoopDB) benefited from automatically selecting hash- or sort-based aggregation depending on the number of groups per rows. A hand coded MapReduce job will most likely have only one of these strategies hard coded and not choose optimal strategies.

1.3.5 Join Task

The join task needed to be hand coded for HadoopDB too due to an implementation bug in Hive. The equivalent SQL is:

```
SELECT sourceIP, COUNT(pageRank), SUM(pageRank),
SUM(adRevenue) FROM Rankings AS R, UserVisits AS UV
WHERE R.pageURL = UV.destURL AND
UV.visitDate BETWEEN '2000-01-15' AND '2000-01-22'
GROUP BY UV.sourceIP;
```

In this task HadoopDB performs ten times faster then Hadoop but still significantly slower then the parallel databases. The databases (including HadoopDB) are said to benefit from a secondary index[6] on visitDate, the selection predicate. There are however at least three questions to be asked about this task.

First, why would anybody want to run this query only for the specified weeks and not for all weeks? If this query would be run for all weeks then the databases would also need to scan all data and maybe approach the duration of Hadoop. Hadoop on the other hand would typically run a nightly run over its data to produce small daily statistics which in turn could be easily imported into a traditional relational database.

Second, as already noted, the UserVisits table hardly resembles a typical server log. A typical server log would be ordered by visitDate. If the visitDate would be the primary key, then Hadoop should be able to execute the query much faster.

Third, since the UserVisits table already contains data from other sources (adRevenue, countryCode, languageCode, searchWord), why isn't there also a column for page rank?

[6]Hive has also implemented support for secondary indices in the meanwhile.

This would of course duplicate data and break traditional database design but better match real world scenarios. In this case the join wouldn't be needed and the task would be mostly the same as the selection task already discussed.

1.3.6 UDF Aggregation Task

This task represents the prime example of MapReduce. The data set consists of a large corpus of HTML documents. Those are parsed and the occurrence frequency of every url gets counted. Not only was it difficult to implement this task with the parallel databases, they also performed significantly worse then Hadoop. HadoopDB wasn't used with it's SQL layer but also queried by MapReduce.

1.3.7 Fault tolerance and heterogeneous environment

All discussed tasks so far have been executed optimistically without any precautions for single node failures. To test the different performance of the systems in the presence of failures and in a heterogeneous environment the aggregation task with grouping on the prefixed sourceIP has been repeated. This time the replication factor was set to 2 and for the fault tolerance test one random node was killed when half of the query was executed. The heterogeneous environment was tested in another round by slowing done one node by running an I/O intensive background job and frequent clears of the OS caches.

The performance loss is reported in relation to the normal performance. Vertica suffers a dramatic slowdown of more then 100% in both cases since it restarts queries from the beginning on failure and does not move straggling tasks to other nodes. HadoopDB and Hadoop both profit from Hadoops inherent design for frequent failures. First Hadoop does not restart the whole query from scratch when one node fails but only that part of the query that the failed node performed. Second Hadoop runs second instances of straggling tasks on already finished nodes. This is called speculative execution. The later started tasks may finish earlier then tasks that run on faulty nodes with degraded performance.

HadoopDB slightly outperforms Hadoop in the case of failing nodes. Hadoop is slowed down in this case because it is eager to create additional copies of the replicated data to provide the replication factor again as soon as possible. HadoopDB on the other hand does not copy data and would therefore suffer a data loss in case that two nodes containing the same data would fail.

1.4 Summary and Discussion

The HadoopDB authors conclude that HadoopDB would outperform Hadoop in all but the last tasks and that it would therefore be a useful tool for large scale analytics workloads.

The comments and discussions in the preceding subsections however challenge this conclusion. The long time to load data may be a problem. Some of the provided benchmarks are rather unrealistic. The number of nodes tested is still a lower limit for typical MapReduce installations. The usage of Hadoop may not have been optimal as seen in the discussion about compression. An experienced Hadoop engineer may be able to achieve better results with the given tasks. The fault tolerance of HadoopDB lacks the self healing mechanism of HDFS. And HadoopDB is designed to be a read only database parallel to the production databases. This seems to be a contradiction to the authors own remark that it would be "a requirement to perform data analysis inside of the DBMS, rather than pushing data to external systems for analysis". [ABPA+09, 7.1]

If HadoopDB would be a useful contribution then it should probably have raised discussions, comments or recommendations on the internet. From this perspective however,

HadoopDB seems to have failed. Google reports[7] less then ten independent mentions of HadoopDB not counting the HadoopDB authors own publications.

Since HadoopDB uses PostgreSQL and provides a solution to cluster this DBMS, one should expect some interest from the PostgreSQL community. There are however only a total of 15 mails in the projects list archive mentioning HadoopDB from which only 4 are from last year.[8]

The public subversion repository of HadoopDB on SourceForge has only 22 revisions.[9]

Although Hadoop is one of the most discussed award winning free software projects right now, HadoopDB did not manage to profit from this and raise any significant interest at all outside of academic circles. This may be one indicator for the correctness of critiques presented in this paper.

An additional critique from a practical perspective may be the combination of two highly complicate systems, Hadoop and PostgreSQL. Both require advanced skills from the administrator. It is hard enough to find personal that can administrate one of these systems. It may be practically impossible to find personal competent in both. Both systems come with their own complexities that add up for little or no benefit.

As already mentioned, Stonebraker already concluded that new databases must be easy to use because hardware is cheaper then personal. The HadoopDB guide however expects sophisticated manual work:[10]

> "a careful analysis of data and expected queries results in a significant performance boost at the expense of a longer loading. Two important things to consider are data partitioning and DB tuning (e.g. creating indices). Again, details of those techniques are covered in textbooks. For some queries, hash-partitioning and clustered indices improve HadoopDB's performance 10 times"

2 Hive

Hive has been developed by Facebook as a petabyte scale data warehouse on top of Hadoop. [TSJ+09] A typical use case for Hive is to store and analyze incrementally (daily) inserted log data. It provides user interfaces and APIs that allow the submission of queries in a SQL like language called HiveQL. The application then parses the queries to a directed-acyclic graph (DAG) of Map Reduce jobs and executes them using Hadoop.

2.1 Data Model

Data in Hive is organized and stored in a three level hierarchy:

- The first level consists of *tables* analogous to those in relational databases. Each table has a corresponding HDFS directory where it stores its data.
- Inside tables the data is further split into one or more *partitions*. Partitioning is done by the different values of the partitioning columns which each gets their own directory inside the table directory. Each subsequent partitioning level leads to an additional sub directory level inside its parent partition.
- Data can be further split in so called buckets. The target bucket is choosen by a hash function over a column modulo the desired number of buckets.[11] Buckets are

[7] as of April 7th, searching for the term "HadoopDB"
[8] as of April 8th `http://search.postgresql.org/search?m=1`
[9] as of June 14th `http://hadoopdb.svn.sourceforge.net/viewvc/hadoopdb`
[10] `http://hadoopdb.sourceforge.net/guide` (2011-06-20)
[11] `http://wiki.apache.org/hadoop/Hive/LanguageManual/DDL/BucketedTables` (2011-06-21)

representes as files in HDFS either in the table directory if there are no partitions or in the deepest partition directory.

The following is an example path of a bucket in the table "daily_status" with partitions on the columns "ds" and "ctry". The hive workspace is in the "wh" directory:

$$/wh/\underbrace{daily\_status}_{\text{table name}}/\underbrace{ds = 20090101/ctry = US}_{\text{nested partitions}}/\underbrace{0001}_{\text{bucket}}$$

Buckets are mostly useful to prototype and test queries on only a sample of the full data. This gives the user a meaningful preview whether a query is correct and could provide the desired information in a small fraction of the time it would take to run the query over the full data set. For this purpose Hive provides a SAMPLE clause that lets the user specify the number of buckets that should be used for the query.

Although no reference has been found for this thesis, buckets may also be useful to distribute the write and read load to multiple HDFS data nodes. Assuming that partitions are defined on date and country as in the path example above and the daily traffic data of web servers in the US should be imported into Hive. In the absence of buckets this import would sequentially write one file after the other to HDFS. Only one HDFS data node would be written to[12] at any given time while the other nodes remaining idle. If the bucketing function is choosen so that nearby data rows are likely to end up in different buckets then the write load gets distributed to multiply data nodes.

Columns in Hive can have the usual primitive scalar types including dates. Additionally Hive also provides nestable collection types like arrays and maps. Furthermore it is possible to program user defined types.

2.2 Query Language

HiveQL looks very much like SQL.[13] Recent versions of Hive even support Views[14] and Indices[15]. A design goal of Hive was to allow users to leverage their existing knowledge of SQL to query data stored in Hadoop.

However Hive can not modify already stored data. This was a design decision to avoid complexity overhead. Thus there are no UPDATE or DELETE statements in HiveQL. [TSJ+10] Regular INSERTs can only write to new tables or new partitions inside a table. Otherwise a INSERT OVERWRITE must be used that will overwrite already existing data. However since a typical use case adds data to hive once a day and starts a new partition for the current date there is no practical limitation caused by this.

For extensibility Hive provides the ability for user defined (aggregation) functions. One example provided is a Python function used to extract memes from status updates.

A noteworthy optimization and addition to regular SQL is the ability to use one SELECT statement as input for multiple INSERT statements. This allows to minimize the number of table scans:

```
FROM (SELECT ...) subq1
INSERT TABLE a SELECT subq1.xy, subq1.xz
INSERT TABLE b SELECT subq1.ab, subq1.xy
```

[12] plus the few replica nodes for the current data block

[13] This ticket tracks tracks the work to minimize the difference between HiveQL and SQL: `https://issues.apache.org/jira/browse/HIVE-1386` (2011-06-21)

[14] since October 2010, `http://wiki.apache.org/hadoop/Hive/LanguageManual/DDL#Create.2BAC8-Drop_View` (2011-06-21)

[15] since March 2011, `http://wiki.apache.org/hadoop/Hive/LanguageManual/DDL#Create.2BAC8-Drop_Index` (2011-06-21)

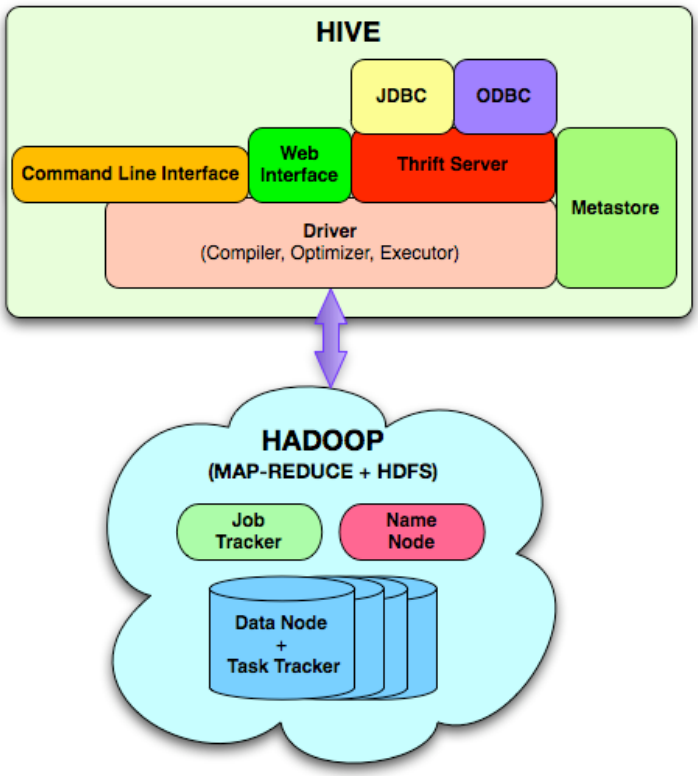

Figure 3: Hive architecture

2.3 Architecture

Hive consists of several components to interact with the user, external systems or programming languages, a metastore and a driver controlling a compiler, optimizer and executor. (Figure 3)

The metastore contains the schema information for tables and information how data in this tables should be serialized and deserialized (SerDe information). This information can be overridden for individual partitions. This allows schemes and serialization formats to evolve without the necessity to rewrite older data. It is also planned to store statistics for query optimization in the metastore. [16]

The compiler transforms a query in multiple steps to an execution plan. For SELECT queries these steps are:

1. Parse the query string into a parse tree.
2. Build an internal query representation, verify the query against schemes from the metastore, expand SELECT * and check types or add implicit type conversions.
3. Build a tree of logical operators (logical plan).
4. Optimize the logical plan.[17]
5. Create a DAG of MapReduce jobs.

[16] The hive paper suggests in its introduction, that the metastore would already store statistics which is later contradicted in section 3.1 Metastore. More information about the ongoing work for statistics is available from the Hive wiki and in the jira issue HIVE-33: `http://wiki.apache.org/hadoop/Hive/StatsDev` (2011-06-21) `https://issues.apache.org/jira/browse/HIVE-33` (2011-06-21)

[17] The optimizer currently supports only rule but not cost based optimizations. The user can however provide several kinds of hints to the optimizer.

References

[ABPA+09] Abouzeid, Azza ; Bajda-Pawlikowski, Kamil ; Abadi, Daniel J. ; Rasin, Alexander ; Silberschatz, Avi: HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In: *PVLDB* 2 (2009), Nr. 1, 922-933. http://www.vldb.org/pvldb/2/vldb09-861.pdf

[CDG+06] Chang, Fay ; Dean, Jeffrey ; Ghemawat, Sanjay ; Hsieh, Wilson C. ; Wallach, Deborah A. ; Burrows, Mike ; Chandra, Tushar ; Fikes, Andrew ; Gruber, Robert E.: Bigtable: a distributed storage system for structured data. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*. Berkeley, CA, USA : USENIX Association, 2006 (OSDI '06), 15-15

[DD08] D. DeWitt, M. S.: *MapReduce: A major step backwards*. http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards, 01 2008. – accessed 11-April-2011

[FHH09] Fanara, Andrew ; Haines, Evan ; Howard, Arthur: The State of Energy and Performance Benchmarking for Enterprise Servers. In: Nambiar, Raghunath (Hrsg.) ; Poess, Meikel (Hrsg.): *Performance Evaluation and Benchmarking* . Berlin, Heidelberg : Springer-Verlag, 2009. – ISBN 978–3–642–10423–7, Kapitel Performance Evaluation and Benchmarking, S. 52–66

[PD10] Peng, Daniel ; Dabek, Frank: Large-scale incremental processing using distributed transactions and notifications. In: *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. Berkeley, CA, USA : USENIX Association, 2010 (OSDI'10), 1-15

[PPR+09] Pavlo, A. ; Paulson, E. ; Rasin, A. ; Abadi, D. J. ; DeWitt, D. J. ; Madden, S. ; Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: *SIGMOD '09* (2009)

[SMA+07] Stonebraker, Michael ; Madden, Samuel ; Abadi, Daniel J. ; Harizopoulos, Stavros ; Hachem, Nabil ; Helland, Pat: The End of an Architectural Era (It's Time for a Complete Rewrite). In: Koch, Christoph (Hrsg.) ; Gehrke, Johannes (Hrsg.) ; Garofalakis, Minos N. (Hrsg.) ; Srivastava, Divesh (Hrsg.) ; Aberer, Karl (Hrsg.) ; Deshpande, Anand (Hrsg.) ; Florescu, Daniela (Hrsg.) ; Chan, Chee Y. (Hrsg.) ; Ganti, Venkatesh (Hrsg.) ; Kanne, Carl-Christian (Hrsg.) ; Klas, Wolfgang (Hrsg.) ; Neuhold, Erich J. (Hrsg.): *33rd International Conference on Very Large Data Bases*. Vienna, Austria : ACM Press, sep 2007. – ISBN 978–1–59593–649–3, S. 1150–1160

[TSJ+09] Thusoo, Ashish ; Sarma, Joydeep S. ; Jain, Namit ; Shao, Zheng ; Chakka, Prasad ; Anthony, Suresh ; Liu, Hao ; Wyckoff, Pete ; Murthy, Raghotham: Hive- A Warehousing Solution Over a Map-Reduce Framework. In: *Proceedings of the VLDB Endowment 2, Nr. 2*, 2009, S. 1626–1629

[TSJ+10] Thusoo, Ashish ; Sarma, Joydeep S. ; Jain, Namit ; Shao, Zheng ; Chakka, Prasad ; 0002, Ning Z. ; Anthony, Suresh ; Liu, Hao ; Murthy, Raghotham: Hive - a petabyte scale data warehouse using Hadoop. In: Li, Feifei (Hrsg.) ; Moro, Mirella M. (Hrsg.) ; Ghandeharizadeh, Shahram (Hrsg.) ; Haritsa, Jayant R. (Hrsg.) ; Weikum, Gerhard (Hrsg.) ; Carey, Michael J. (Hrsg.) ; Casati, Fabio (Hrsg.) ; Chang, Edward Y. (Hrsg.) ; Manolescu, Ioana

(Hrsg.) ; Mehrotra, Sharad (Hrsg.) ; Dayal, Umeshwar (Hrsg.) ; Tsotras, Vassilis J. (Hrsg.): *ICDE*, IEEE, 03 2010 (Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA). – ISBN 978–1–4244–5444–0, S. 996–1005