Data Warehousing and Analytics Infrastructure at Facebook

Ashish Thusoo
Zheng Shao
Suresh Anthony

Dhruba Borthakur
Namit Jain
Joydeep Sen Sarma

Raghu Murthy
Hao Liu

Facebook[1]

[1] The authors can be reached at the following addresses:
{athusoo,dhruba,rmurthy,zshao,njain,hliu,
suresh,jssarma}@facebook.com

ABSTRACT

Scalable analysis on large data sets has been core to the functions of a number of teams at Facebook - both engineering and non-engineering. Apart from ad hoc analysis of data and creation of business intelligence dashboards by analysts across the company, a number of Facebook's site features are also based on analyzing large data sets. These features range from simple reporting applications like Insights for the Facebook Advertisers, to more advanced kinds such as friend recommendations. In order to support this diversity of use cases on the ever increasing amount of data, a flexible infrastructure that scales up in a cost effective manner, is critical. We have leveraged, authored and contributed to a number of open source technologies in order to address these requirements at Facebook. These include Scribe, Hadoop and Hive which together form the cornerstones of the log collection, storage and analytics infrastructure at Facebook. In this paper we will present how these systems have come together and enabled us to implement a data warehouse that stores more than 15PB of data (2.5PB after compression) and loads more than 60TB of new data (10TB after compression) every day. We discuss the motivations behind our design choices, the capabilities of this solution, the challenges that we face in day today operations and future capabilities and improvements that we are working on.

Categories and Subject Descriptors

H.m [**Information Systems**]: Miscellaneous.

General Terms

Management, Measurement, Performance, Design, Reliability, Languages.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD'10, June 6–10, 2010, Indianapolis, Indiana, USA.

Copyright 2010 ACM 978-1-4503-0032-2/10/06...$10.00.

Keywords

Data warehouse, scalability, data discovery, resource sharing, distributed file system, Hadoop, Hive, Facebook, Scribe, log aggregation, analytics, map-reduce, distributed systems.

1. INTRODUCTION

A number of applications at Facebook rely on processing large quantities of data. These applications range from simple reporting and business intelligence applications that generate aggregated measurements across different dimensions to the more advanced machine learning applications that build models on training data sets. At the same time there are users who want to carry out ad hoc analysis on data to test different hypothesis or to answer one time questions posed by different functional parts of the company. On any day about 10,000 jobs are submitted by the users. These jobs have very diverse characteristics such as degree of parallelism, execution time, resource needs and data delivery deadlines. This diversity in turn means that the data processing infrastructure has to be flexible enough to support different processing needs - both from the perspective of supporting different SLAs as well as from the perspective of supporting optimal algorithms and techniques for the different query patterns.

What makes this task even more challenging is the fact that the data under consideration continues to grow rapidly as more and more users end up using Facebook as a ubiquitous social network and as more and more instrumentation is added to the site. As an example of this tremendous data growth one has to just look at the fact that while today we load between 10-15TB of compressed data every day, just 6 months back this number was in the 5-6TB range. Note that these sizes are the sizes of the data after compression – the uncompressed raw data would be in the 60-90TB range (assuming a compression factor of 6). Needless to say, such a rapid growth places very strong scalability requirements on the data processing infrastructure. Strategies that are reliant on systems that do not scale horizontally are completely ineffective in this environment. The ability to scale using commodity hardware is the only cost effective option that enables us to store and process such large data sets.

In order to address both of these challenges – diversity and scale, we have built our solutions on technologies that support these characteristics at their core. On the storage and compute side we rely heavily on Hadoop[1] and Hive[2] – two open source technologies that we have significantly contributed to, and in the case of Hive a technology that we have developed at Facebook. While the former is a popular distributed file system and map-reduce platform inspired by Google's GFS[4] and map-reduce[5] infrastructure, the later brings the traditional data warehousing tools and techniques such as SQL, meta data, partitioning etc. to the Hadoop ecosystem. The availability of such familiar tools has tremendously improved the developer/analyst productivity and created new use cases and usage patterns for Hadoop. With Hive, the same tasks that would take hours if not days to program can now be expressed in minutes and a direct consequence of this has been the fact that we see more and more users using Hive and Hadoop for ad hoc analysis in Facebook – a usage pattern that is not easily supported by just Hadoop. This expanded usage has also made data discovery and collaboration on analysis that much more important. In the following sections we will also touch upon some systems that we have built to address those important requirements.

Just as Hive and Hadoop are core to our storage and data processing strategies, Scribe[3] is core to our log collection strategy. Scribe is an open source technology created at Facebook that acts as a service that can aggregate logs from thousands of web servers. It acts as a distributed and scalable data bus and by combining it with Hadoop's distributed file system (HDFS)[6] we have come up with a scalable log aggregation solution that can scale with the increasing volume of logged data.

In the following sections we present how these different systems come together to solve the problems of scale and job diversity at Facebook. The rest of the paper is organized as follows. Section 2 describes how the data flows from the source systems to the data warehouse. Section 3 talks about storage systems, formats and optimizations done for storing large data sets. Section 4 describes some approaches taken towards making this data easy to discover, query and analyze. Section 5 talks about some challenges that we encounter due to different SLA expectations from different users using the same shared infrastructure. Section 6 discusses the statistics that we collect to monitor cluster health, plan out provisioning and give usage data to the users. We conclude in Section 7.

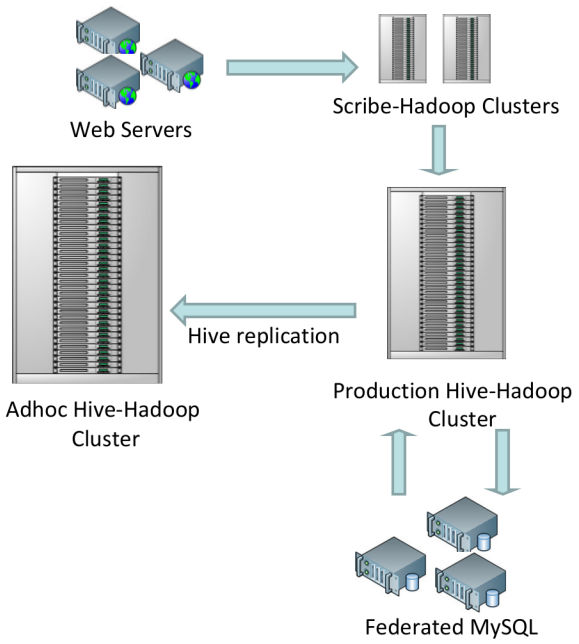

Figure 1: Data Flow Architecture

2. DATA FLOW ARCHITECTURE

In **Error! Reference source not found.** we illustrate how the data flows from the source systems to the data warehouse at Facebook. As depicted, there are two sources of data – the federated mysql tier that contains all the Facebook site related data and the web tier that generates all the log data. An example of a data set that originates in the former includes information describing advertisements – their category, there name, the advertiser information etc. The data sets originating in the latter mostly correspond to actions such as viewing an advertisement, clicking on it, fanning a Facebook page etc. In traditional data warehousing terminology, more often than not the data in the federated mysql tier corresponds to dimension data and the data coming from the web servers corresponds to fact data.

The data from the web servers is pushed to a set of Scribe-Hadoop (scribeh) clusters. These clusters comprise of Scribe servers running on Hadoop clusters. The Scribe servers aggregate the logs coming from different web servers and write them out as HDFS files in the associated Hadoop cluster. Note that since the data is passed uncompressed from the web servers to the scribeh clusters, these clusters are generally bottlenecked on network. Typically more than 30TB of data is transferred to the scribeh clusters every day – mostly within the peak usage hours. In order to reduce the cross data center traffic the scribeh clusters are located in the data centers hosting the web tiers. While we are exploring possibilities of compressing this data on the web tier before pushing it to the scribeh cluster, there is a trade off between compression and the latencies that can be introduced as a result of it, especially for low volume log categories. If the log category does not have enough volume to fill up the compression buffers on the web tier, the data therein can experience a lot of delay before it becomes available to the users unless the compression buffer sizes are reduced or periodically flushed. Both of those possibilities would in turn lead to lower compression ratios.

Periodically the data in the scribeh clusters is compressed by copier jobs and transferred to the Hive-Hadoop clusters as shown in Figure 1. The copiers run at 5-15 minute time intervals and copy out all the new files created in the scribeh clusters. In this manner the log data gets moved to the Hive-Hadoop clusters. At this point the data is mostly in the form of HDFS files. It gets published either hourly or daily in the form of partitions in the corresponding Hive tables through a set of loader processes and then becomes available for consumption.

The data from the federated mysql tier gets loaded to the Hive-Hadoop clusters through daily scrape processes. The scrape processes dump the desired data sets from mysql databases, compressing them on the source systems and finally moving them

into the Hive-Hadoop cluster. The scrapes need to be resilient to failures and also need to be designed such that they do not put too much load on the mysql databases. The latter is accomplished by running the scrapes on a replicated tier of mysql databases thereby avoiding extra load on the already loaded masters. At the same time any notions of strong consistency in the scraped data is sacrificed in order to avoid locking overheads. The scrapes are retried on a per database server basis in the case of failures and if the database cannot be read even after repeated tries, the previous days scraped data from that particular server is used. With thousands of database servers, there are always some servers that may not be reachable by the scrapes and by a combination of using retries and scraping stale data a daiy dump of the dimension data is created in the Hive-Hadoop clusters. These dumps are then converted to top level Hive tables.

As shown in Figure 1, there are two different Hive-Hadoop clusters where the data becomes available for consumption by the down stream processes. One of these clusters – the production Hive-Hadoop cluster - is used to execute jobs that need to adhere to very strict delivery deadlines, where as the other cluster – the ad hoc Hive-Hadoop cluster is used to execute lower priority batch jobs as well as any ad hoc analysis that the users want to do on historical data sets. The ad hoc nature of user queries makes it dangerous to run production jobs in the same cluster. A badly written ad hoc job can hog the resources in the cluster, thereby starving the production jobs and in the absence of sophisticated sandboxing techniques, the separation of the clusters for ad hoc and production jobs has become the practical choice for us in order to avoid such scenarios.

Some data sets are needed only in the ad hoc cluster where as others are needed in both the clusters. The latter are replicated from the production cluster to the ad hoc cluster. A Hive replication process checks for any changes made to the Hive tables in the production cluster. For all changed tables, the replication process copies over the raw data and then reapplies the meta data information from the production Hive-Hadoop cluster to the ad hoc Hive-Hadoop cluster. It is important to load the data to the production cluster first as opposed to the ad hoc cluster both from the point of view of making the data available earlier to the critical production jobs and also from the point of view of not putting the less reliable ad hoc cluster on the path of the data arriving at the production cluster. The replication process relies on the logging of all the Hive commands submitted to the production cluster. This is achieved by implementing a "command logger" as a pre-execution hook in Hive – the pre-execution hook API enables Hive users to plug in any kind of programming logic that gets executed before the execution of the Hive command.

Finally, these published data sets are transformed by user jobs or queried by ad hoc users. The results are either held in the Hive-Hadoop cluster for future analysis or may even be loaded back to the federated mysql tier in order to be served to the Facebook users from the Facebook site. As an example all the reports generated for the advertisers about their advertisement campaigns are generated in the production Hive-Hadoop cluster and then loaded to a tier of federate mysql data bases and served to the advertisers from the advertiser Insights web pages.

2.1 Data Delivery Latency

As is evident from this discussion, the data sets arrive in the Hive-Hadoop clusters with latencies ranging from 5-15 minutes in case of logs to more than a day in case of the data scraped from the federated mysql databases. Moreover, even though the logs are available in the form of raw HDFS files within 5-15 minutes of generation (unless there are failures), the loader processes loads the data into native Hive tables only at the end of the day. In order to give users more immediate access to these data sets we use Hive's external table feature[7] to create table meta data on the raw HDFS files. As the data gets loaded into the native tables, the raw HDFS files are removed and thus get purged from the external tables. By running the loaders daily we are able to compact the data to fit in smaller number of files while still providing user access to the most recent data through external tables on the raw HDFS files.

In general this scheme has worked well for us so far, but going forward there are requirements to reduce latencies even further, especially for applications that want to incrementally build indexes or models that are time critical. At the same time in order to scale with the increasing amount of data, it has also become important for us to move towards continuously loading log data into our clusters, so that bandwidth and cluster resources can be used in a more incremental manner as opposed to a bulk manner – which is what happens today when the copiers are executed at 15 minute time intervals. Similarly for scrapes, as the size of our database tier increases we are working on solutions to incrementally scrape those databases both to reduce the load on the databases and also to reduce the data delivery latencies for the dimension data in Hive.

3. STORAGE

With such large incoming data rates and the fact that more and more historical data needs to be held in the cluster in order to support historical analysis, space usage is a constant constraint for the ad hoc Hive-Hadoop cluster. The production cluster usually has to hold only one month's worth of data as the periodic production jobs seldom look at data beyond that time frame. However, the ad hoc cluster does need to hold all the historical data so that measures, models and hypotheses can be tested on historical data and compared with the recent data. Due to the huge quantity of data involved, we keep all our data compressed using gzip. Hadoop allows us the ability to compress data using a user specified codecs. For the majority of our data sets we rely on the gzip codec. We get a compression factor of 6-7 on most of our data sets.

In addition to using gzip we have also developed and deployed row columnar compression in Hive for many of our tables. This compression scheme is based on PAX[8] storage layout for data records. With this scheme we have seen between 10-30% reduction in space requirements when compared with using plain gzip on our data sets stored in Hadoop SequenceFiles (a file format that supports storing binary data in HDFS). Moreover, this compression has resulted in slight improvement in CPU utilization on some of our benchmarks with marked improvements on queries that use a small subset of table columns.

Though better space utilization through better compression techniques and in some cases domain specific compression techniques are obvious, a major inefficiency in HDFS that can be addressed to tremendously reduce the storage footprint is the fact that 3 copies of the same is stored by default in HDFS, in order to prevent data loss due to node failures. However, by using erasure codes this multiple can be brought down to 2.2 - by storing two copies of the data and 2 copies of the error correction codes for the data[10]. This does reduce the copies of the raw data and thus can reduce data locality for user jobs as Hadoop tries to execute tasks on the nodes that hold the corresponding data and by reducing the number of nodes that hold the copy of the data the likelihood of the task running on a node that does not have a copy of the data, increases. However, in our cluster we hardly see the network as a bottleneck and with 100MB/s bandwidth between any two nodes that are located on the same rack, executing tasks on racks that have a local copy of the data is sufficient to ensure that network does not become a bottleneck for user jobs. We are still experimenting with erasure codes or Hadoop RAID and we have seen a lot of positive results by using it on our test data sets. Moreover, if parallelism does become an issue, with Hadoop we have the option of applying this technique on a per data set partition basis. We can enable Hadoop RAID on older data sets which are not accessed by too many jobs while keeping the newer data sets replicated 3 ways.

3.1 Scaling HDFS NameNode

Though raw storage can be addressed through compression techniques and through using RAID for error correction, another significant resource contraint that we have encountered in our clusters is the memory used by the HDFS NameNode. The NameNode is the master server that holds the file to block mappings for HDFS and as the number of files and the number of blocks in the system increases, the memory required to hold all this state on the NameNode also increases. Currently we have close to 100 million blocks and files on the ad hoc Hive-Hadoop cluster and the NameNode heap size is configured to use 48GB of memory. Apart from more and more optimizations on the NameNode data structures, we are also exploring simple techniques like creating archive files by concatenating a number of HDFS files together in order to reduce the number of files in the file system. With tools like Hive and SQL the explosion of files is further compounded primarily because it is very easy to write jobs that can produce outputs that is fragmented over a large number of small sized files. As an example a simple filter query in Hive when run over a large data set would create the same number of files as the number of map tasks used to scan the input data set. However, if the filter is extremely selective, most of these files would end up being empty and this would end up creating a very large number of small files in HDFS, putting even more memory pressure on the NameNode. Moreover, if this Hive query was periodically run every day – as is often the case – the problem would just get worse. In order to address such cases we have enhanced Hive to produce plans which include a concatenation step for such queries, trading off some bit of job latency for the reduced pressure on the NameNode memory. We have also implemented the HiveCombinedFileInputFormat which also significantly reduces the number of map tasks needed by downstream jobs in case the data set is already fragmented across a large number of files. With this input format, a single map task executing on a node is able to read blocks from different files that are stored on that node – a capability that is not available on the default input formats in Hadoop.

3.2 Federation

With the continued growth in our data sets at some point in the near future we will have to seriously consider federating data across multiple Hadoop clusters housed in different data centers. In general there are two approaches that we have considered – distributing data across clusters on the basis of time so that older data sets are held in clusters that have more storage than compute, or distributing data across clusters on the basis of application groups e.g. separating the ads data sets from other data sets in a separate cluster of its own. Both approaches have their pros and cons. While the benefits of the first one are easy to understand, and to some extent techniques used in hierarchical storage management can be applied to manage these data sets, the fact that the data sets have been growing rapidly make its benefits less clear. Rapid and compounded growth means that much more data is newer and that separation of old data into another cluster would end up saving only 20-25% space on the existing cluster. Additionally it also means more user intervention in case data sets across the "time boundary" have to be compared or joined. On the other hand, federating on the basis of applications has the down side that some of the common dimension data would have to be replicated and that would mean more storage overhead. At this time though, we think that this overhead is very low and by federating on the basis of applications we can balance out the storage and query workloads on different clusters more evenly.

4. DATA DISCOVERY AND ANALYSIS

At Facebook querying and analysis of data is done predominantly through Hive. The data sets are published in Hive as tables with daily or hourly partitions. The primary interfaces to interact with Hive for ad hoc query purposes are a web based GUI - HiPal - and the Hive command line interface – Hive CLI. In addition a home grown job specification framework called Databee is used for specifying job and data dependencies. In this section we discuss in more detail these tools and systems that are used heavily to find, query and analyze data sets in Hive.

4.1 Hive

Hive[8] is a data warehousing framework built on top of Hadoop. It was created at Facebook and then contributed back to the Hadoop ecosystem as a Hadoop subproject. Inside Facebook it is used heavily for reporting, ad hoc querying and analysis. The basic motivation behind the creation of Hive was the observation that while Hadoop promised scalability and while map/reduce was a good lowest common denominator to express diverse kinds of jobs, it was too low level to be used to develop pipelines in a productive manner. In addition, SQL and the concepts of tables, columns and partitions are concepts and tools that many users are familiar with. As a result it was natural to conclude that by putting structure on top of the data in Hadoop and by providing SQL as a tool to query that data, the power of this platform could be brought to the masses. Without Hive, the same job would take hours if not days to author in map-reduce. Using Hive the task could be expressed very easily in a matter of minutes. It has been

possible with Hive to bring the immense scalability of map-reduce to the non engineering users as well – business analysts, product managers and the like who, though familiar with SQL would be in a very alien environment if they were to write map-reduce programs for querying and analyzing data.

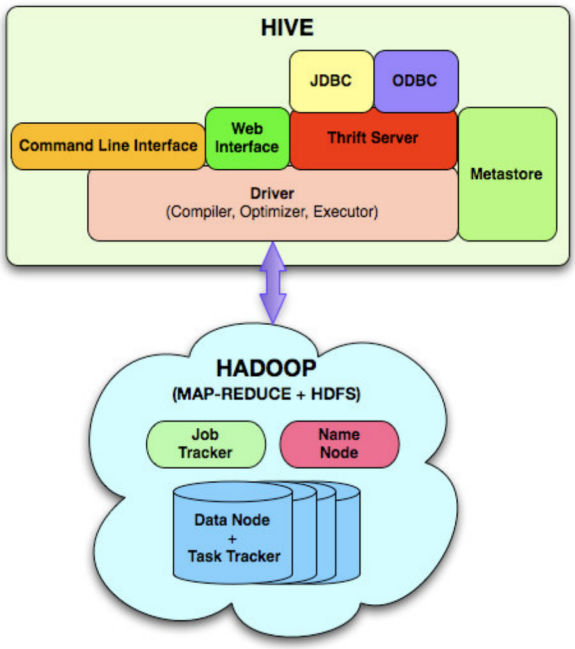

Figure 2: Hive System Architecture

Figure 2 shows the major components of Hive and how it interacts with Hadoop. The tables and partitions in Hive are stored as HDFS directories and these mappings and the structural information of these objects are stored in the Hive Metastore. The Driver uses this information to convert transformations expressed in HiveQL(Hive query language - a SQL variant) into a sequence of map-reduce jobs and HDFS operations. A number of optimizations are employed during this compilation phase. These optimizations include regular optimizations like pruning files to be scanned on the basis of the partitioning predicates specified in the query, pushing down of predicates so that data filtering happens immediately after the scan so that as few a rows as possible flow through the operator tree and across map-reduce jobs and finally pruning of columns which are not used by the query, again with the intention of reducing the amount of data flowing through the query plan. In addition to these optimizations, other optimizations needed to generate efficient plans in the presence of data skew are also done by the optimizer e.g. a group by computation done on a data set with a skewed distribution on the group by key is done through two map-reduce jobs – the first job for generating partial aggregates and the second job for generating complete aggregates from these partial aggregates. In addition a number of optimizations to do partial aggregations in the map tasks also help in reducing the amount of data crossing the map-reduce boundary. A number of different join techniques such as reordering join orders so that the larger tables are streamed in the reduce tasks and not held in memory, map side joins for increased parallelism in the case where one of the join tables is small enough to fit in the memory of a map task and bucketed joins that employ the knowledge of how the data is hashed into the HDFS files of a table and many other optimizations, also help in creating efficient plans which only an expert map-reduce programmer could encode into map-reduce programs. All these lead to not just big productivity gains or the users but also lead to more efficient usage of the cluster resources.

Apart from increasing developer productivity and cluster efficiency, Hive is also extensible in many different ways. This flexibility enables it to be used for transformations that cannot be easily expressed in SQL. By allowing users to plug in their own computation through user defined functions, user defined table functions or by specifying their own custom scripts with a TRANSFORM/MAP/REDUCE keywords in HiveQL, Hive provides a truly flexible way of expressing different types of transformations. In addition it provides interfaces that allow users to specify their own types or data formats which becomes very useful while dealing with legacy data or rich data types.

All these capabilities have been instrumental in making it the core system for doing analysis – both ad hoc as well as periodic – on large data sets at Facebook.

4.2 Interactive Ad hoc queries

At Facebook ad hoc queries are executed by the users through either HiPal, a web based graphical interface to Hive or Hive CLI a command line interface similar to mysql shell. HiPal specially enables users who are not very familiar with SQL to graphically create queries for ad hoc analysis. It supports capabilities to show job progress, inspect results, upload and download results and small data sets in CSV format and capabilities to report and debug errors. In addition the results of any query are stored in the cluster for 7 days so that they can be shared amongst users. HiPal is an extremely popular tool which is used heavily in Facebook for authoring and executing ad hoc queries. While it has truly democratized the ability to query large data sets, it has also lead to novel usage patterns and use cases for Hadoop, use cases that were not core to the original set of applications that it was designed for. As an example the ability to submit bad queries has exposed deficiencies in Hadoop when it comes to protecting jobs from side effects of poorly written queries. It has also opened interesting scheduling questions around how to ensure that long running periodic jobs in the cluster do not starve the smaller ad hoc jobs of resources and poor user experience for users executing ad hoc queries. We will discuss these issues around isolation and fair sharing in section 5. Also, in general ad hoc users tend to be more interactive and we are still evolving our systems to support this interactivity on what has traditionally been a batch processing system. Despite these limitations, the ability to be able to query these data sets in an easy and intuitive manner has been the driving force behind the continued adoption and popularity of HiPal throughout Facebook.

4.3 Data Discovery

A direct consequence of the increasing data sets and their diverse uses is the explosion in meta data itself. Today we have more than 20,000 tables on the ad hoc Hive-Hadoop cluster with hundreds of users querying these tables every month. As a result

discovering data is itself challenging. This is further compounded by the fact that there is no central process to author this data as is usually the case with data warehouses in other companies. Decentralization if augmented with the right tools can itself develop into a highly efficient process by itself, which is a must have in a dynamic and fast paced environment like Facebook. As a result we have embraced and experimented with some ideas and concepts around using a combination of machines and people to make data discovery efficient.

We have created internal tools that enable a wiki approach to meta data creation. Users can add and correct table and column descriptions and can also tag tables with searchable tags and project names. The idea here is that users can collaborate on generating this information so that over a period of time this becomes an accurate and searchable corpus of meta data that keeps adapting to schema changes and changes in data sets. Apart from the user generated content we have also written tools to extract lineage information from query logs. This information has been used to build tools that enable users to navigate tables and quickly see the base data sets that have been used to generate any particular data set and also all the data sets derived from a particular table. This capability along with simple tools for inspecting a small set of rows of the table gives a lot of information to the users about the origin and structure of data and aids in data discovery. The query logs have also been a mined to identify expert users on a per table basis. These are users who have frequently queried the associated tables in the recent past. We have also built tools that enable the users to connect to these expert users and ask questions about the data. All these tools are integrated into HiPal and provide the necessary data discovery tools within the context of a query session – right at the time when the user needs these tools the most.

4.4 Periodic Batch Jobs

Apart from ad hoc jobs a large part of the cluster is used by periodic jobs that are run at different periodicities, ranging from 5 minutes to once every week. For such jobs, inter job dependencies and the ability to schedule jobs when certain data sets are available are critical. In addition a common pattern in these jobs involves waiting for certain base data sets, transforming them in one or more steps using Hive and then loading the results to various different mysql database tiers. Moreover such jobs require monitoring and alerting in case of failures and good dashboards to check their current status, past runs or time of completion.

All these capabilities are provided by Databee, which is a python framework for specifying such jobs. It supports different operators for moving data between different systems as well as operators for transforming data or running queries in different clusters. It provides the ability to specify dependencies on data sets or on other Databee jobs and it also generates monitoring information that tracks completion times and latencies for the job and its intermediate steps. In addition it also collects stats on the amount of data handled by different stages in the data pipeline. These stats in some cases are useful to discover bugs in the transformation logic.

Facebook analysts also use Microstrategy for dimensional analysis. In such situations the periodic jobs construct the cubes for Microstrategy and then load them into Microstrategy server instances. The ODBC drivers in Hive enable Microstrategy servers to submit such jobs directly to Hive.

5. RESOURCE SHARING

The co-existence of interactive ad hoc queries and periodic batch jobs on the same set of cluster resources has many implications on how resources are shared between different jobs in the cluster. Ad hoc users require minimal response times from the system and the ability to tweak and change the jobs as they refine their queries. On the other hand the periodic batch jobs require a predictable execution time/latency as they are more concerned with data being available before a certain deadline. The deadline is what comprises of an SLA between the authors of these jobs and the downstream consumers of the data that they produce. The goals of experimentation for ad hoc users and predictability for periodic batch jobs are at odds with each other. At the same time since a job in Hadoop cannot be pre-empted to give up its resources, there are scenarios where a long running periodic job can deprive a short running interactive ad hoc query of cluster resources thereby significantly increasing the latency for such jobs. These opposing requirements have been central in shaping our strategy on how resources are shared and allocated to different Hadoop jobs.

In order to support co-existence of interactive jobs and batch jobs on the same Hadoop cluster, Facebook has been instrumental in the development of the Hadoop Fair Share Scheduler[10] and we use it in all our clusters. Users are divided into pools and cluster resources are shared equally among pools. Each user in the system gets his/her own pool. We set limits on the number of concurrent jobs and tasks per pool. The periodic jobs are run in their own separate special pools. These types of special pools have minimum resource quotas associated with them and the jobs that use these pools are likely to get at least their minimum quota even when there is a high load average on the cluster. These techniques have played a critical part in supporting both interactive ad hoc jobs and periodic batch jobs on our clusters. There are still challenges especially since we have not enabled pre-emption of tasks on our clusters. A direct consequence of not having pre-emption is that starvation is still an issue as fair sharing is done only for new task allocations and does not actually kill tasks for jobs which have already used up their fair share.

In order to isolate the cluster from poorly written queries – which can happen quite easily because of ad hoc queries supported by our clusters – we have made improvements to the Hadoop Fair Share Scheduler to make it more aware of system resources, - primarily the memory and CPU consumed by each job. Before these improvements a common scenario in our ad hoc cluster would involve a user submitting a bad query whose tasks would consume a lot of memory on a majority of nodes in the cluster. As a result these machines would start swapping at best and would even crash in the worst case. Either scenario would have an adverse side effect on all the jobs running on the cluster, including the periodic batch jobs that require a lot of predictability in their run times. Needless to say, this would soon snowball into a big problem for the users. In order to prevent such scenarios from playing out on the cluster, we have improved

the Fair Share Scheduler to actively monitor the CPU and memory usage of all the nodes in the cluster. The scheduler kills existing tasks on a node if memory usage on that node exceeds configured thresholds. Moreover, the scheduler allocates new tasks to a slave node only if the CPU and memory usage of that node is within configured limits. This improvement in the scheduler has gone a long way in isolating our clusters from bad jobs. The allocations still do not take into account network usage and disk I/O usage per task, but we have seen that those resources are less of an issue in our deployments. Apart from these improvements, we have also enabled the locality wait feature in the scheduler. This feature delays scheduling a task - up to a certain period of time - till it finds a node with a free task slot that also contains a copy of the data that is needed by the task. However, this strategy does not work well for jobs working on small data sets – which we also find a lot in our ad hoc cluster. For those small jobs, we do not use locality wait. Moreover, for such jobs the map and reduce tasks are scheduled simultaneously instead of scheduling the reduce tasks only after a fraction of the map tasks are completed.

As mentioned previously, we execute periodic batch jobs that have to deliver data to very strict deadlines in a separate production cluster. This simple method of separation further ensures that ad hoc jobs that take up too many resources cannot impact the execution times of critical production jobs in any way. Predictability of execution times is extremely critical for these jobs and this method further isolates these jobs from the unpredictable patterns of the ad hoc queries. Physically separating the clusters does mean that we may not be getting the maximum utilization from our hardware – there are times when the nodes in the ad hoc cluster are idle while there are plenty of jobs pending in the queue of the other cluster and vice versa. We are experimenting with a system called the "dynamic clouds" where in we can decommission nodes from one cluster and move them over to another cluster on demand. This movement of nodes is only across map-reduce clusters and not across HDFS cluster as it is easy to move compute resources but very difficult to move storage resources, and it is usually the former that is needed for the jobs that have accumulated in the cluster queue. This is still an experimental system which is still being developed.

6. OPERATIONS

There have been a lot of lessons that we have learnt while operating the data warehousing and analytics infrastructure at Facebook. With such a huge deployment and with vast parts of the company depending on this infrastructure for analysis needs – including parts that run business critical jobs on this infrastructure – it is natural that we devote a lot of our energies on the operational aspects of these deployments. Broadly speaking, operations comprises of two things – supporting the systems that this infrastructure is built on and supporting the data pipelines that run on this infrastructure. We collect a lot of statistics for these purposes, which we will talk about in this section.

We have added a lot of monitoring to the nodes in our clusters and the Hadoop daemon processes. We collect regular system level stats such as CPU usage, I/O activity, memory usage etc. from all the nodes. In addition to these we also track Hadoop specific statistics both for the JobTracker (which is the master for the map-reduce cluster) and the NameNode (which is the master for the HDFS cluster). From the JobTracker we collect statistics like average job submission rate, number of utilized map and reduce slots, job tracker JVM heap utilization as well as the frequency of garbage collection in the job tracker. From the NameNode, apart from the statistics related to the JVM heap and garbage collection we also monitor the space usage as well as the "balancedness" of the cluster – the later comprising of a measure of standard deviation for the distribution of the space used across the nodes in the HDFS cluster. In addition we also track the count of bad blocks in HDFS as well as the number of slave nodes in both the map-reduce cluster and the HDFS cluster. These measures help us to quickly identify problems in the cluster and take any remedial action. We have hooked these up with our internal systems to generate appropriate alerts in case these measures cross certain thresholds. Apart from these measures which are critical in identifying problems in the cluster, there are others that we regularly collect for figuring out our provisioning needs. These include the HDFS space usage and the aggregated amount of CPU and memory being used in the cluster. Moreover, we have also developed simple tools like htop – a top like utility which gives CPU and memory usage across the entire cluster on a per job basis. This has been very helpful in identifying jobs that take up inordinate amount of system resources in the cluster.

Along with system level monitoring, we also provide a number of dashboards to our users to provide them feedback on the space and compute that they are using on the cluster. We track the space usage by users and teams as well as the growth rate of the different data sets. We regularly track tables or data sets that suffer from fragmentation - those tables that store a small data set in a large number of files. We also provide ways in which users and teams can set up retention limits on their data sets in Hive, so that their table partitions can be automatically cleaned up. Going forward we are adding more and more capabilities to give us more intelligence on how the cluster is used and how this usage can be further optimized.

For users that run periodic batch jobs we also provide capabilities to monitor the run time and failures for their jobs. Databee collects a lot of this information and hooks them up with our monitoring systems. We also track the publish time for a number of our data sets so that consumers for those data sets can be alerted in case of delays. This enables us to get a better end to end picture of our entire infrastructure and helps us identify weak links or under provisioned tiers/clusters involved in the data flow. We constantly measure the uptimes of our clusters and track how we are doing against our SLAs to our users and we are constantly looking at ways and measures to improve the user experience for our users.

7. CONCLUSION

A lot of different components come together to provide a comprehensive platform for processing data at Facebook. This infrastructure is used for various different types of jobs each having different requirements – some more interactive in nature as compared to others, some requiring a predictable execution time as compared to others that require the ability to experiment and tweak. This infrastructure also needs to scale with the

tremendous amount of data growth. By leveraging and developing a lot of open source technologies we have been able to meet the demands placed on our infrastructure and we are working on many other enhancements to it in order to service those demands even more and in order to evolve this infrastructure to support new use cases and query patterns.

8. ACKNOWLEDGMENTS

The current state of the data warehousing and analytics infrastructure has been the result of on going work over the last couple of years. During this time a number of people at Facebook have made significant contributions and enhancements to these systems. We would like to thank Scott Chen, Dmytro Molkov and Rodrigo Schmidt for contributing a number of enhancements to Hadoop - including htop, resource aware scheduling, dynamic clouds, Hadoop RAID etc. We would also like to thank He Yongqiang, John Sichi, Ning Zhang, Paul Yang and Prasad Chakka for a number of contributions to Hive. These include features like row column compression, table functions, various join optimizations, views, Hive metastore etc. Also thanks are due to Andrew Ryan and Rama Ramasamy for doing a lot of work on operations, monitoring and the statistics setup that makes these tasks easy. Thanks are also due to Anthony Giardullo, Gautam Roy and Aron Rivin for their continued support and enhancements to Scribe and for helping make scribe-hdfs integration possible. Vijendar Ganta has been instrumental in building a number of web based tools including the collaboration features in HiPal. HiPal itself owes its genesis to a Facebook hackathon project contributed to by David Wei and Ravi Grover amongst others. Acknowledgements are also due to Matei Zaharia for implementing the Fair Share Scheduler as part of his internship at Facebook and the open source Hadoop and Hive communities that have contributed immensely to both these projects. Last but not the least thanks are also due to the users of our infrastructure who have patiently dealt with periods of instability during its evolution and have provided valuable feedback that enabled us to make continued improvements to this infrastructure. They have constantly kept the bar high for us and have had a major contribution in the evolution of this infrastructure.

9. REFERENCES

[1] Apache Hadoop wiki. Available at http://wiki.apache.org/hadoop.

[2] Apache Hadoop Hive wiki. Available at http://wiki.apache.org/hadoop/Hive.

[3] Scribe wiki. Available at http://wiki.github.com/facebook/scribe.

[4] Ghemawat, S., Gobioff, H. and Leung, S. 2003. The Google File System. In Proceedings of the 19th ACM Symposium on Operating Systems Principles (Lake George, NY, Oct. 2003).

[5] Dean, J. and Ghemawat S. 2004. MapReduce: Simplified Data Processing on Large Clusters. In Proceedings of the 6th Symposium on Operating System Design and Implementation (San Francisco, CA, Dec. 2004). OSDI'04.

[6] HDFS Architecture. Available at http://hadoop.apache.org/common/docs/current/hdfs_design.pdf.

[7] Hive DDL wiki. Available at http://wiki.apache.org/hadoop/Hive/LanguageManual/DDL.

[8] Thusoo, A., Murthy, R., Sen Sarma, J., Shao, Z., Jain, N., Chakka, P., Anthony, A., Liu, H., Zhang, N. 2010. Hive – A Petabyte Scale Data Warehouse Using Hadoop. In Proceedings of 26th IEEE International Conference on Data Engineering (Long Beach, California, Mar. 2010). ICDE'10.

[9] Ailamaki, A., DeWitt, D.J., Hill, M.D., Skounakis, M. 2001. Weaving Relations for Cache Performance. In Proceedings of 27th Very Large Data Base Conference (Roma, Italy, 2001). VLDB'01.

[10] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I. 2009. Job Scheduling for Multi-User MapReduce Clusters. UC Berkeley Technical Report UCB/EECS-2009-55 (Apr. 2009).

[11] Fan, B., Tantisiriroj, W., Xiao, Lin, Gibson, G. 2009. DiskReduce: RAID for Data-Intensive Scalable Computing. In Proceedings of 4th Petascale Data Storage Workshop Supercomputing Conference (Portland, Oregon, Nov. 2009). Supercomputing PDSW'09.