Spring 2016

Data Migration from RDBMS to Hadoop

Naga Sruthi Tiyyagura
Governors State University

Monika Rallabandi
Governors State University

Radhakrishna Nalluri
Governors State University

Table of Contents

1 ProjectDescription

1.1 Project Abstract

Oracle, IBM, Microsoft and Teradata own a large portion of the information on the planet. By that on the off chance that we run an inquiry in any piece of the world, it is likely that you are perusing the information from a Database possessed by them. The bigger the volume of information moves from Oracle to DB2 or other is testing assignment for the business. The conception of Hadoop and NoSQL innovation spoke to a seismic movement that shook the RDBMS market and offering a different option for organizations. The Database merchants moved rapidly to Big Data for position and opposite. Indeed, even everybody has own enormous information innovation like prophet NoSQL and mongo DB ,There is a colossal business sector for an elite information movement that can duplicate the information and put away in RDBMS Databases to Hadoop or NoSQL databases. Current data is available in the RDBMS databases like oracle, SQL Server, MySQL and Teradata. We are planning to migrate RDBMS data to big data which is support NoSQL database and contains verity of data from the existed system it's take huge resources and time to migrate pita bytes of data. Time and resource may be constraints for the current migrating process.

1.2 Competitive Information

In the summary report "Big Data Best Practices", we see 45 different examples of big data exploitation by businesses and industries of all sorts: entertainment, manufacturing, retail, financial, food services, travel, sports, fashion, politics, gaming, and much more. This report refers to the now famous 2011 McKinsey Global Institute report (Big Data: The Next Frontier for Innovation, Competition, and Productivity) when making this declaration: "there are no [Big Data] best practices. I'd say there are emerging next practices." Consequently, it is now time for all businesses to get on board that train and exploit big data for competitive advantage. The white paper "25 Data Stories from GNIP" provides another rich compilation of stories that demonstrate the "unlimited value and near limitless application" of big data, with a focus on business growth and competitive advantage through social data exploitation.

- Hundreds of built-in data-type conversions, transformers, look-up matching, and aggregations
- Robust metadata, data lineage, and data modeling capabilities
- Data quality and profiling subsystems
- Workflow management, i.e., a GUI for generating ETL scripts and handling errors
- Fine grained, role-based security

1.3 Relationship to Other Applications/Projects

Normally folks use NoSQL DBs(like HBase, Cassandra) with Hadoop. Using these DBs with hadoop is merely a matter of configuration. You don't need any connecting program in order to achieve this. Apart from the point made by @Doctor Dan, there are few other reasons behind choosing NoSQL DBs in place of SQL DBs. One thing is size. These NoSQL DBs provided great horizontal scalability which enables you to store PBs of data easily. You could scale traditional systems, but vertically. Another reason for complexityof data. The places, where these DBs are being used, mostly handle highly unstructured data which is not very easy to deal with using traditional systems. For example, sensor data, log data etc. Basically, I did not understand why sqoop exists. Why can't we directly use an SQL data on Hadoop.

Although Hadoop is very good at handling your BigData needs, it is not the solution to all your needs. It is not suitable for real-time needs. Suppose you are an Online Transaction Company with very very huge dataset. You find out that you could process this data very easily using Hadoop. But the problem is that you can't serve the real-time needs of you customers with Hadoop. This is where SQOOP comes into picture. It is an import/export tool that allows you to move data between a SQL DB and Hadoop. You could move your BigData into your Hadoop cluster, process it there and then push the results back into your SQL DB using SQOOP to serve the real-time needs of your customers.

A traditional RDBMS is used to handle relational data. Hadoop works well with structured as well as unstructured data, and supports various serialization and data formats for example Text, Json, Xml, Avro etc. I would say there are problems where SQL databases are a perfect choice. if your data size permits it and your data type is relational, you are fine to use the RDBMS approach. Its worked well in the past, its a mature technology and it has its needs. Where the data size or type is such that you are unable to save it in an RDBMS, go for solutions like Hadoop. One such example is a product catalog. A car has different attributes than a television. It is tough to create a new table per product type. Another example is machine generated data. in this case the data size creates a big pressure on the traditional RDBMS. Thats a classic Hadoop problem. Or document indexing. There are various such examples.

1.4 Assumptions and Dependencies

In CDH 3, all of the Hadoop API implementations were confined to a single JAR file (hadoop-core) plus a few of its dependencies. It was relatively straightforward to make sure that classes from these JAR files were available at runtime.

CDH 4 and CDH 5 are more complex: they bundle both MRv1 and MRv2 (YARN). To simplify things, CDH 4 and CDH 5 provide a Maven-based way of managing client-side Hadoop API dependencies that saves you from having to figure out the exact names and locations of all the JAR files needed to provide Hadoop APIs.

In CDH 5, Cloudera recommends that you use a hadoop-client artifact for all clients, instead of managing JAR-file-based dependencies manually.

- Flavors of the hadoop-client Artifact
- Versions of the hadoop-client Artifact
- Using hadoop-client for Maven-based Java Projects
- Using hadoop-client for Ivy-based Java Projects
- Using JAR Files Provided in the hadoop-client Package

1.5 Future Enhancements

Hadoop is immature technology. As such, it naturally offers much room for improvement in both industrial-strengthens and performance. And since Hadoop is booming, multiple efforts are underway to fill those gaps. For example:

- Cloudera’s proprietary code is focused on management, set-up, etc.
- The “Phase 1″ plans Hortonworks shared with me for Apache Hadoop are focused on industrial-strengthness, as are significant parts of “Phase 2″.*
- MapR tells a performance story versus generic Apache Hadoop HDFS and MapReduce. (One aspect of same is just C++ vs. Java.)
- So does Hadapt, but mainly vs. Hive.
- Cloudera also tells me there’s a potential 4-5X performance improvement in Hive coming down the pike from what amounts to an optimizer rewrite.

(Zettaset belongs in the discussion too, but made an unfortunate choice of embargo date.)

Hortonworks, a new Hadoop company spun out of Yahoo, graciously permitted me to post a slide deck outlining an Apache Hadoop roadmap. Phase 1 refers to stuff that is underway more or less now. Phase 2 is scheduled for alpha in October, 2011, with production availability not too late in 2012.

You’ve probably heard some single point of failure fuss. Hadoop NameNodes can crash, which wouldn’t cause data loss, but would shut down the cluster for a little while. It’s hard to come up with real-life stories in which this has been a problem; still, it’s something that should be fixed, and everybody (including the Apache Hadoop folks, as part of Phase has a favored solution. A more serious problem is that Hadoop is currently bad forsmall updates, because:

- Hadoop’s fundamental paradigm assumes batch processing.
- Both major workarounds to allow small updates are broken:
    - HBase is seriously buggy, to the point that it sometimes loses data.
    - Storing each update in a separate file runs afoul of a practical limit of 70-100 million files.

File-count limits also get blamed for a second problem, in that there may not be enough intermediate files allowed for your Reduce steps, necessitating awkward and perhaps poorly-performing MapReduce workarounds. Anyhow, the Phase 2 Apache Hadoop roadmap features a serious HBase rewrite. I’m less clear as to where things stand with respect to file-count limits.

Edits: As per the comments below, I should perhaps have referred to HBase’s HDFS underpinnings rather than HBase itself. Anyhow, some details are in the slides. Please also see my follow-up post on *how well HBase is indeed doing*.

The other big area for Hadoop improvement is modularity, pluggability, and coexistence, on both the storage and application execution tiers. For example:

- Greenplum/MapR and Hadapt both think you should have HDFS file management and relational DBMS coexisting on the same storage nodes. (I agree.)
- Part of what Hortonworks calls “Phase 2″ sets out to ensure that Hadoop can properly manage temp space and so on next to HDFS.
- Perhaps HBase won’t always assume HDFS.
- DataStax thinks you should blend HDFS and Cassandra.

Meanwhile, Pig and Hive need to come closer together. Often you want to stream data into Hadoop. The argument that MPI trumps MapReduce does, in certain use cases, make sense. Apache Hadoop “Phase 2″ and beyond are charted to accommodate some of those possibilities too.

1.6 Definitions and Acronyms

ETL.. Ectract, Transform and Load
SQL.. Structured Query Language
HDFS.. Hadoop Distributed File System
XML.. Extensible Markup Language
GUI.. Graphical User Interphase
HSQLDB.. Hyper SQL Database
NFS.. Network File System
CLI.. Command Line
API.. Application Programming Interphases
JDBC.. Java Database Connectivity
ODBC..Open Database Connectivity
SIEM.. Security Information Event Management
Vm.. Virtual Machine
Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs.

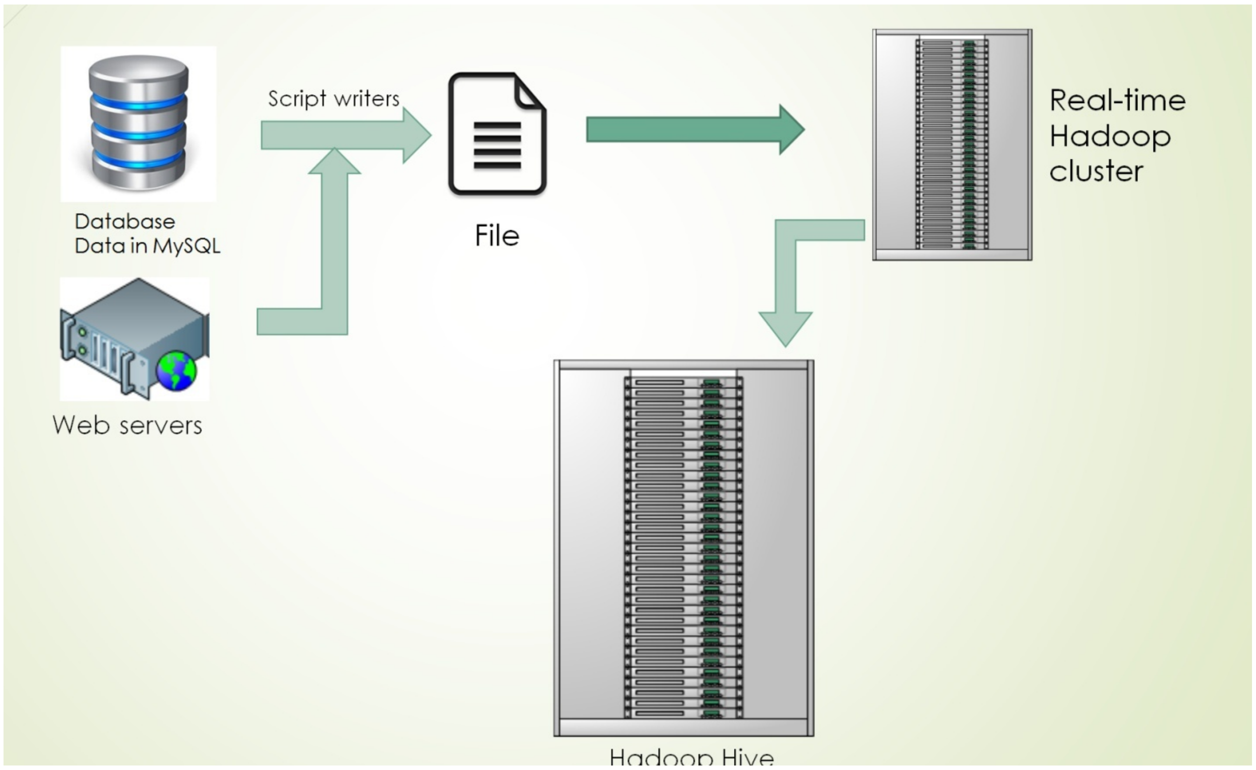
- **Open-source software.** Open-source software is created and maintained by a network of developers from around the globe. It's free to download, use and contribute to, though more and more commercial versions of Hadoop are becoming available.
- **Framework.** In this case, itmeans that everything you need to develop and run software applications is provided – programs, connections, etc.

- **Massive storage.** The Hadoop framework breaks big data into blocks, which are stored on clusters of commodity hardware.
- **Processing power.** Hadoop concurrently processes large amounts of data using multiple low-cost computers for fast results.

2 Technical Description

By utilizing Sqoop we will import information from a social database framework into HDFS. The info to the import procedure is a database table. Sqoop will read the table column by-line into HDFS. The yield of this import procedure is an arrangement of documents containing a duplicate of the foreign made table. The import procedure is performed in parallel. Thus, the yield will be in different documents. These documents may be delimited content records or paired .A by-result of the import procedure is a created Java class which can epitomize one line of the foreign made table. This class is utilized amid the import process by Sqoop itself.In the wake of controlling the foreign records with Hive we will have outcome information set which you can then fare back to the social database. Sqoop's fare procedure will read an arrangement of delimited content documents from HDFS in parallel, parse them into records, and supplement them as new lines in an objective database table, for utilization by outer applications or clients.

2.1 Project/Application Architecture



2.2 Project/Application Information flows:

Application Information flows have three steps strategy to move data from RDMS to HIVE on Hadoop.
Flows have three steps strategy to move data from RDMS to HIVE on Hadoop.
Depending on whether direct access is available to the RDBMS source system, you may opt for either a File Processing method (when no direct access is available) or RDBMS Processing (when database client access is available).

Regardless of the ingest option, the processing workflow in this article requires:

1. One-time, initial load to move all data from source table to HIVE.

2. On-going, “Change Only” data loads from the source table to HIVE.

Below, both File Processing and Database-direct (SQOOP) ingest will be discussed.

2.3 File Processing

Step 1: Converting RDMS Data into Hadoop

For this blog, we assume that a file or set of files within a folder will have a delimited format and will have been generated from a relational system (i.e. records have unique keys or identifiers).

Files will need to be moved into HDFS using standard ingest options:

- WebHDFS: Primarily used when integrating with applications, a Web URL provides an Upload end-point into a designated HDFS folder.
- NFS: Appears as a standard network drive and allows end-users to use standard Copy-Paste operations to move files from standard file systems into HDFS.

Once the initial set of records are moved into HDFS, subsequent scheduled events can move files containing only new Inserts and Updates.

SQOOP is the JDBC-based utility for integrating with traditional databases. A SQOOP Import allows for the movement of data into either HDFS (a delimited format can be defined as part of the Import definition) or directly into a Hive table.

Step 2: Store file into Hadoop cluster

In the background, the source file is split into HDFS blocks, the size of which is configurable (commonly 128 MB, 64 MB by default). For fault tolerance, each block is automatically replicated by HDFS. By default, three copies of each block are written to three different DataNodes. The replication factor is user-configurable (default is three). The DataNodes are servers which are physical machines or virtual machines/cloud instances. DataNodes form the Hadoop cluster into which you write your data and on which you run your MapReduce/Hive/Pig/Impala/Mahout/etc. programs.TheDataNodes are the workers of the Hadoop cluster, the NameNodes are the masters.

When a file is to be written into HDFS, the client writing the file obtains from the NameNode a list of DataNodes that can host replicas of the first block of the file.The client arranges a pipeline through which all bytes of data from the first block of the source file will be transmitted to all participating DataNodes. The pipeline is formed from client to first DataNode to second DataNode to final (third in our case) DataNode. The data is split into packets for transmission, and each packet is tracked until all DataNodes return acks to indicate successful replication of the data. The packets are streamed to the first DataNode in the pipeline, which stores the packet and forwards it to the second DataNode, and so on. If one or more replications fail, the infrastructure automatically constructs a new pipeline and retries the copy.

When all three DataNodes confirm successful replication, the client will advance to the next block, again request a list of host DataNodes from the NameNode, and construct a new pipeline. This process is followed until all blocks have been copied into HDFS. The final block written may be smaller than the configured block size, but all blocks from the first to the penultimate block will be of the configured block size.

Step 3: Read Data from HIVE

The tables in Hive are similar to tables in a relational database, and data units are organized in a taxonomy from larger to more granular units. Databases are comprised of tables, which are made up of partitions. Data can be accessed via a simple query language and Hive supports overwriting or appending data.

Within a particular database, data in the tables is serialized and each table has a corresponding Hadoop Distributed File System (HDFS) directory. Each table can be sub-divided into partitions that determine how data is distributed within sub-directories of the table directory. Data within partitions can be further broken down into buckets.

Hive supports all the common primitive data formats such as BIGINT, BINARY, BOOLEAN, CHAR, DECIMAL, DOUBLE, FLOAT, INT, SMALLINT, STRING, TIMESTAMP, and TINYINT. In addition, analysts can combine primitive data types to form complex data types, such as structs, maps and arrays.

2.4 Interactions with other Applications

Hive is a data warehousing infrastructure built on top of apache Hadoop.

Hadoop provides massive scale-out and fault-tolerance capabilities for data storage and processing (using the MapReduce programming paradigm) on commodity hardware.

Hive enables easy data summarization, ad-hoc querying and analysis of large volumes of data.

It is best used for batch jobs over large sets of immutable data (like web logs).

It provides a simple query language called Hive QL, which is based on SQL and which enables users familiar with SQL to easily perform ad-hoc querying, summarization and data analysis.

At the same time, Hive QL also allows traditional MapReduce programmers to be able to plug in their custom mappers and reducers to do more sophisticated analysis that may not be supported by the built-in capabilities of the languag

Hive Query Language capabilities:

Hive query language provides the basic SQL like operations. These operations work on tables or partitions.

- Ability to create and manage tables and partitions (create, drop and alter).
- Ability to support various Relational, Arithmetic and Logical Operators.
- Ability to do various joins between two tables.
- Ability to evaluate functions like aggregations on multiple “group by” columns in a table.
- Ability to store the results of a query into another table.
- Ability to download the contents of a table to a local directory.
- Ability to create an external table that points to a specified location within HDFS
- Ability to store the results of a query in an HDFS directory.
- Ability to plug in custom scripts using the language of choice for custom map/reduce jobs.

Major Components of Hive and its interaction with Hadoop:

Hive provides external interfaces like command line (CLI) and web UI, and application programming interfaces (API) like JDBC and ODBC. The Hive Thrift Server exposes a very simple client API to execute HiveQL statements. Thrift is a framework for cross-language services, where a server written in one language (like Java) can also support clients in other languages.

The Metastore is the system catalog. All other components of Hive interact with the Metastore.

The Driver manages the life cycle of a HiveQL statement during compilation, optimization and execution.

The Compiler is invoked by the driver upon receiving a HiveQL statement. The compiler translates this statement into a plan which consists of a DAG of map/reduce jobs.

The driver submits the individual map/reduce jobs from the DAG to the Execution Engine in a topological order. Hive currently uses Hadoop as its execution engine.

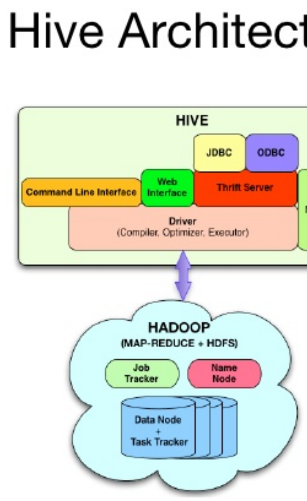
What Hive is NOT

Hive is not designed for online transaction processing and does not offer real-time queries and row-level updates.

Hive aims to provide acceptable (but not optimal) latency for interactive data browsing, queries over small data sets or test queries.

Hive Applications:

- Log processing
- Text mining
- Document indexing
- Customer-facing business intelligence (e.g., Google Analytics)
- Predictive modeling, hypothesis testing



Data is the new currency of the modern world. Businesses that successfully maximize its value will have a decisive impact on their own value and on their customers' success. As the de-facto platform for big data, Apache Hadoop allows businesses to create highly scalable and cost-efficient data stores. Organizations can then run massively parallel and high-performance analytical workloads on that data, unlocking new insight previously hidden by technical or economic limitations. Hadoop offers data value at unprecedented scale and efficiency -- in part thanks to Apache Tez and YARN.

Analytic applications perform data processing in purpose-driven ways that are unique to specific business problems or vendor products. There are two prerequisites to creating purpose-built applications for Hadoop data access. The first is an "operating system" (somewhat akin to Windows or Linux) that can host, manage, and execute these applications in a shared Hadoop environment. Apache YARN is that data operating system for Hadoop. The second prerequisite is an application-building framework and a common standard that developers can use to write data access applications that run on YARN.

Apache Tez meets this second need. Tez is an embeddable and extensible framework that enables easy integration with YARN and allows developers to write native YARN applications that bridge the spectrum of interactive and batch workloads. Tez leverages Hadoop's unparalleled ability to process petabyte-scale datasets, allowing projects in the Apache Hadoop ecosystem to express fit-to-purpose data processing logic, yielding fast response times and extreme throughput. Tez brings unprecedented speed and scalability to Apache projects like Hive and Pig, as well as to a growing field of third-party software applications designed for high-speed interaction with data stored in Hadoop.

Hadoop in a post-MapReduce world

Those familiar with MapReduce will wonder how Tez is different. Tez is a broader, more powerful framework that maintains MapReduce's strengths while overcoming some of its limitations. Tez retains the following strengths from MapReduce:

- Horizontal scalability with increasing data size and compute capacity
- Resource elasticity to work both when capacity is abundant and when it's limited

- Fault tolerance and recovery from inevitable and common failures in distributed systems
- Secure data processing using built-in Hadoop security mechanisms

But Tez is not an engine by itself. Rather, Tez provides common primitives for building applications and engines -- thus, its flexibility and customizability. Developers can write MapReduce jobs using the Tez library, and Tez comes with a built-in implementation of MapReduce, which can be used to run any existing MapReduce job with Tez efficiency.

MapReduce was (and is) ideal for Hadoop users that simply want to start using Hadoop with minimal effort. Now that enterprise Hadoop is a viable, widely accepted platform, organizations are investing to extract the maximum value from data stored in their clusters. As a result, customized applications are replacing general-purpose engines such as MapReduce, bringing about greater resource utilization and improved performance.

The Tez design philosophy

Apache Tez is optimized for such customized data-processing applications running in Hadoop. It models data processing as a data flow graph, so projects in the Apache Hadoop ecosystem can meet requirements for human-interactive response times and extreme throughput at petabyte scale. Each node in the data flow graph represents a bit of business logic that transforms or analyzes data. The connections between nodes represent movement of data between different transformations.

Once the application logic has been defined via this graph, Tez parallelizes the logic and executes it in Hadoop. If a data-processing application can be modeled in this manner, it can likely be built with Tez. Extract-Transform-Load (ETL) jobs are a common form of Hadoop data processing, and any custom ETL application is a perfect fit for Tez. Other good matches are query-processing engines like Apache Hive, scripting languages like Apache Pig, and language-integrated, data processing APIs like Cascading for Java and Scalding for Scala.

2.5 Capabilities

As the amount of data continues to grow exponentially, data scientists increasingly need the ability to perform full-fidelity analysis of that data at massive scale. Cloudera, the leader in enterprise analytic data management powered by Apache Hadoop, today announced a number of new initiatives to enable data scientists to take advantage of big data and Hadoop for data analysis with more complex workflows.Beginning with the introduction of Ibis, an open source project incubating within Cloudera Labs, the company is enabling advanced data analysis on a 100% Python stack—bringing a native Python experience to Hadoop at scale. Cloudera has also announced that, as a direct contributor and industry leader in education around Hadoop, Cloudera will be hosting and organizing the first-everWrangle Conference, an event focused exclusively on real-world applications of data science, from the startup to the enterprise.“Hadoop has evolved dramatically over the last decade, from a batch processing tool to an entire ecosystem that powers most of today’s information architecture as well as traditional BI workloads,” said Wes McKinney, a software engineer at Cloudera and the creator of Python pandas. “We want to build on this momentum and make Hadoop’s infrastructure more accessible to the data science community. We’re doing that by bringing Python more fully into the ecosystem and focusing on the real-world, practical applications of data science.”

Ibis

Cloudera recognized the importance of the Python language in modern data engineering and data science and how, thanks to its use of more complex workflows, it has become a primary language for data transformation and interactive analysis. Python development has been confined to local data processing and smaller data sets, requiring data scientists to make many compromises when attempting to work with big data. Using Ibis, a new open source data analysis framework, Python users will finally be able to process data at scale without compromising user experience or performance.

The initial version of Ibis provides an end-to-end Python experience with comprehensive support for the built-in analytic capabilities in Impala for simplified ETL, data wrangling, and analytics. Upcoming versions will allow users to leverage the full range of Python packages as well as express efficient custom logic using Python. By integrating with Impala, the leading MPP database engine for Hadoop, Ibis can achieve the interactive performance and scalability necessary for big data.

“With its usability, extensibility and robust third-party library ecosystem, it’s easy to understand why Python is the open source language of choice for so many data scientists. However, we recognize its limitation – where it’s unable to achieve high performance at Hadoop-scale,” said Wes McKinney. “With Ibis, our vision is to provide a first-class Python experience on large scalable architectures like Hadoop, with full access to the ecosystem of Python tools.

XML processing

While many traditional databases provide XML support, the XML content must first be loaded into the database. Because Hive tables can be linked to a collection of XML files or document fragments stored in the Hadoop file system, Hadoop is much more flexible in analyzing XML content.

Web and text processing

In addition to XPath operators, the Hive query language offers several ways to work with common web and text data. Tableau exposes the following functions that you can use in calculated fields:

- JSON Objects: GET_JSON_OBJECT retrieves data elements from strings containing JSON objects.
- URLs: Tableau offers PARSE_URL to extract the components of a URL such as the protocol type or the host name. Additionally, PARSE_URL_QUERY can retrieve the value associated with a given query key in a key/value parameter list.
- Text Data: The regular expression find and replace functions in Hive are available in Tableau for complex text processing.

On-the-fly ETL

Custom SQL gives you the flexibility of using arbitrary queries for your connection, which allows complex join conditions, pre-filtering, pre-aggregation and more. Traditional databases rely heavily on optimizers, but they can struggle with complex Custom SQL and lead to unexpected performance degradation as you build views. The batch-oriented nature of Hadoop allows it to handle layers of analytical queries on top of complex Custom SQL with only incremental increases to query time.

Because Custom SQL is a natural fit for the complex layers of data transformations seen in ETL, a Tableau connection to Hive based on Custom SQL is essentially on-the-fly ETL. Refer to the Hadoop and Tableau Demo on the Tableau blog to see how you can use Custom SQL to unpivot nested XML data directly in the Custom SQL connection, yielding views built from the unpivoted data.

Initial SQL

Tableau supports initial SQL for Hadoop Hive connections, which allows you to define a collection of SQL statements to perform immediately after the connection is established. For example, you can set Hive and Hadoop configuration variables for a given connection from Tableau to tune performance characteristics. Refer to the Designing for Performancearticle for more information.

Custom analysis with UDFs and Map/Reduce

Similarly, Tableau allows you to take advantage of custom UDFs and UDAFs built by the Hadoop community or by your own development team. Often these are built as JAR files that Hadoop can easily copy across the cluster to support distributed computation. To take advantage of JAR files or scripts, inform Hive of the location of these files and Hive will take care of the rest.

2.6 Risk Assessment and Management

The following were the disadvantages of hadoop:

- As big data is not suitable for small business
- There is a missing encryption methodology for storage and network levels.
- There are lot of stability issues in Hadoop.

3 Project Requirements

3.1 Identification of Requirements

Their creation, called Hive, allows SQL developers to write Hive Query Language (HQL) statements that are similar to standard SQL statements; now you should be aware that HQL is limited in the commands it understands, but it is still pretty useful. HQL statements are broken down by the Hive service into Map Reduce jobs and executed across a Hadoop cluster.

Hive looks very much like traditional database code with SQL access. However, because Hive is based on Hadoop and Map Reduce operations, there are several key differences. The first is that Hadoop is intended for long sequential scans, and because Hive is based on Hadoop, you can expect queries to have a very high latency (many minutes). This means that Hive would not be appropriate for applications that need very fast response times, as you would expect with a database such as DB2. Finally, Hive is read-based and therefore not appropriate for transaction processing that typically involves a high percentage of write operations.

Hive has three main functions: data summarization, query and analysis. It supports queries expressed in a language called HiveQL, which automatically translates SQL-like queries into Map Reduce jobs executed on Hadoop. In addition, HiveQL supports custom Map Reduce scripts to be plugged into queries. Hive also enables data serialization/deserialization and increases flexibility in schema design by including a system catalog called Hive-Metastore.

Hive supports text files (also called flat files), Sequence Files (flat files consisting of binary key/value pairs) and RCFiles (Record Columnar Files which store columns of a table in a columnar database way.)

3.2 Security and Fraud Prevention

Since fraud is so hard to prove in courts, most organizations and individuals try to prevent fraud from happening by blanket measures. This includes limiting the amount of damage the fraudster can impact on the organization as well as early detection of fraud patterns. For example, credit card companies can cut the credit card limit across the board in anticipation of a few negative fraud cases. Advertisers can prevent advertising campaigns with low number of qualifying events. And anti-terrorism agencies can prevent people with bottles of pure water from boarding the planes. These actions are often in contrast with the company efforts to attract more customers and result in general dissatisfaction. To the rescue are new technologies like Hadoop, Influence Diagrams and Bayesian Networks which are computationally expensive (these are NP-hard in computer science terminology) but are more accurate and predictive.Hadoop is known for its gnawing power. Nothing can compare with the throughput power of thousands of machines each of which has multiple cores. As was reported recently at the Hadoop Summit 2010, the largest installations of Hadoop have 2,000 to 4,000 computers with 8 to 12 cores each, amounting to up to 48,000 active threads looking for a pattern at the same time. This allows either (a) looking through larger periods of time to incorporate events across a larger time frame or (b) taking more sources of information into account. It is quite common among social network companies to comb through twitter blogs in search of relevant data.

Finally, one of the fraud prevention problems is latency. The agencies want to react to an event as soon as possible, often within a few minutes of the event. Yahoo recently reported that it can adjust its behavioral model in a response to a user click event within 5-7 minutes across several hundred of millions of customers and billions of events per day. Cloudera has developed a tool, Flume, that can load billions of events into HDFS within a few seconds and analyze them using MapReduce.

Often fraud detection is akin to “finding a needle in a haystack”. One has to go through mountains of relevant and seemingly irrelevant information, build dependency models, evaluate the impact and thwart the fraudster actions. Hadoop helps with finding patterns by processing mountains of information on thousands of cores in a relatively short amount of time.

As fraud and security breaches are becoming more frequent and sophisticated, traditional security solutions are not able to protect company assets. MapR enables organizations to analyze unlimited amounts and types of data in real time, widen the scale and accelerate the speed of threat analysis, and improve risk assessment by building sophisticated machine learning models. Specific use cases include protection against infrastructure risks as well as consumer-oriented risks across different industries:

- Security Information and Event Management (SIEM): Analyze and correlate large amounts of real-time data from network and security devices to manage external and internal security threats, improve incident response time and compliance reporting.
- Application Log Monitoring: Improve analysis of application log data to better manage system resource utilization, security issues, and diagnose and preempt production application problems.
- Network Intrusion Detection: Monitor and analyze network traffic to detect, identify, and report on suspicious activity or intruders.
- Fraud Detection: Use pattern/anomaly recognition on larger volumes and variety of data to detect and prevent fraudulent activities by external or internal parties.

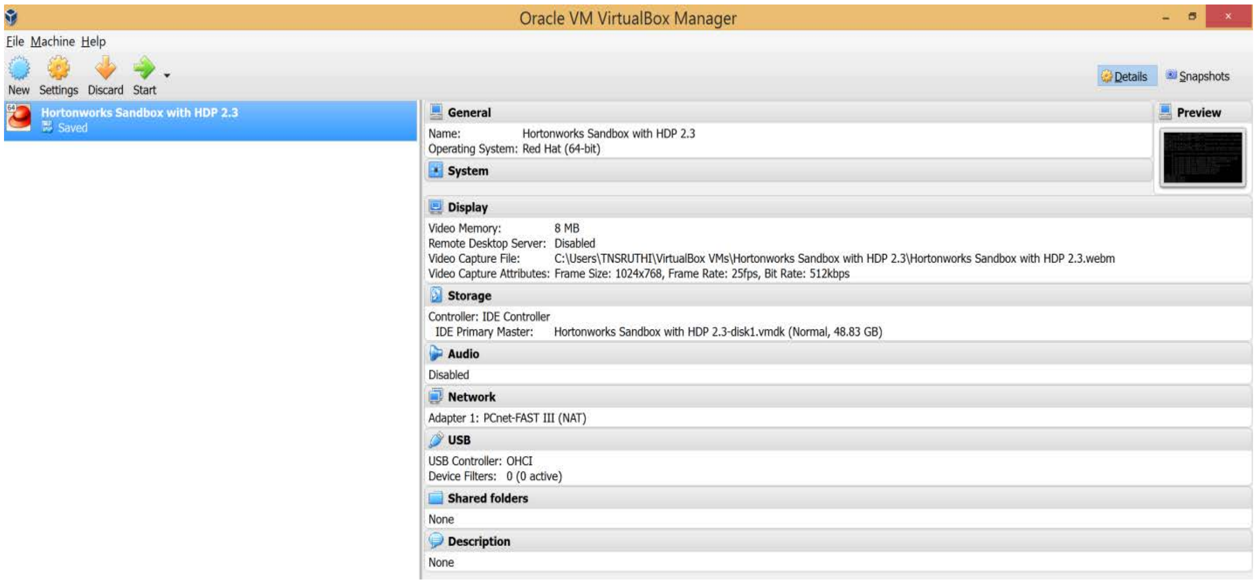
Risk Modeling: Improve risk assessment and associated scoring by building sophisticated machine learning models on Hadoop that can take into account hundreds or even thousands of indicators.

MapR Advantages:

- Easy data ingestion: Copying data to and from the MapR cluster is as simple as copying data to a standard file system using Direct Access NFS™. Applications can therefore ingest data into the Hadoop cluster in real time without any staging areas or separate clusters just to ingest data.
- Existing applications work: Due to the MapR platform's POSIX compliance, any non- Java application works directly on MapR without undergoing code changes. Existing toolsets, custom utilities and applications are good to go on day one.
- Multi-tenancy: Support multiple user groups, any and all enterprise data sets, and multiple applications in the same cluster. Data modelers, developers and analysts can all work in unison on the same cluster without stepping on each other's toes.
- Business continuity: MapR provides integrated high availability (HA), data protection, and disaster recovery (DR) capabilities to protect against both hardware failure as well as site-wide failure.
- High scalability: Scalability is key to bringing all data together on one platform so the analytics are much more nuanced and accurate. MapR is the only platform that scales all the way to a trillion files without compromising performance.
- High performance: The MapR Distribution for Hadoop was designed for high performance, with respect to both high throughput and low latency. In addition, a fraction of servers are required for running MapR versus other Hadoop distributions, leading to architectural simplicity and lower capital and operational expenses.

4 Project Description

Install Oracle VM Virtual Box Manager and load Hortonworks Sandbox with HDP 2.3


Oracle VM VirtualBox Manager
File Machine Help
New Settings Discard Start
Details Snapshots
Hortonworks Sandbox with HDP 2.3
Saved
Preview

General

Name: Hortonworks Sandbox with HDP 2.3
Operating System: Red Hat (64-bit)

System

Display

Video Memory: 8 MB
Remote Desktop Server: Disabled
Video Capture File: C:\Users\TNSRUTHI\VirtualBox VMs\Hortonworks Sandbox with HDP 2.3\Hortonworks Sandbox with HDP 2.3.webm
Video Capture Attributes: Frame Size: 1024x768, Frame Rate: 25fps, Bit Rate: 512kbps

Storage

Controller: IDE Controller
IDE Primary Master: Hortonworks Sandbox with HDP 2.3-disk1.vmdk (Normal, 48.83 GB)

Audio

Disabled

Network

Adapter 1: PCnet-FAST III (NAT)

USB

USB Controller: OHCI
Device Filters: 0 (0 active)

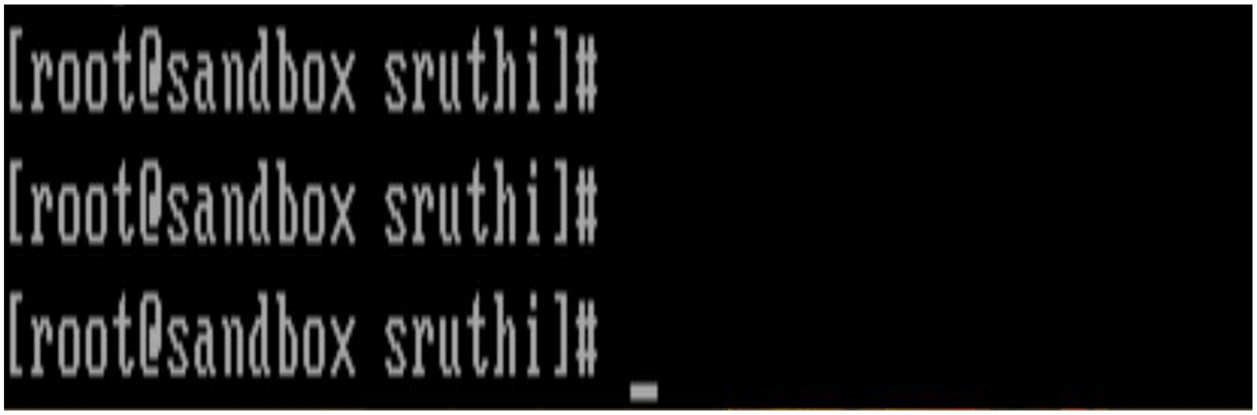
Shared folders

None

Description

None

Connect to the Server:



```
[root@sandbox sruthi]#
[root@sandbox sruthi]#
[root@sandbox sruthi]# _
```

Data flows have three steps strategy to move data from RDMS to HIVE on Hadoop.

Step 1: Convert the all data into files by using Sqoop

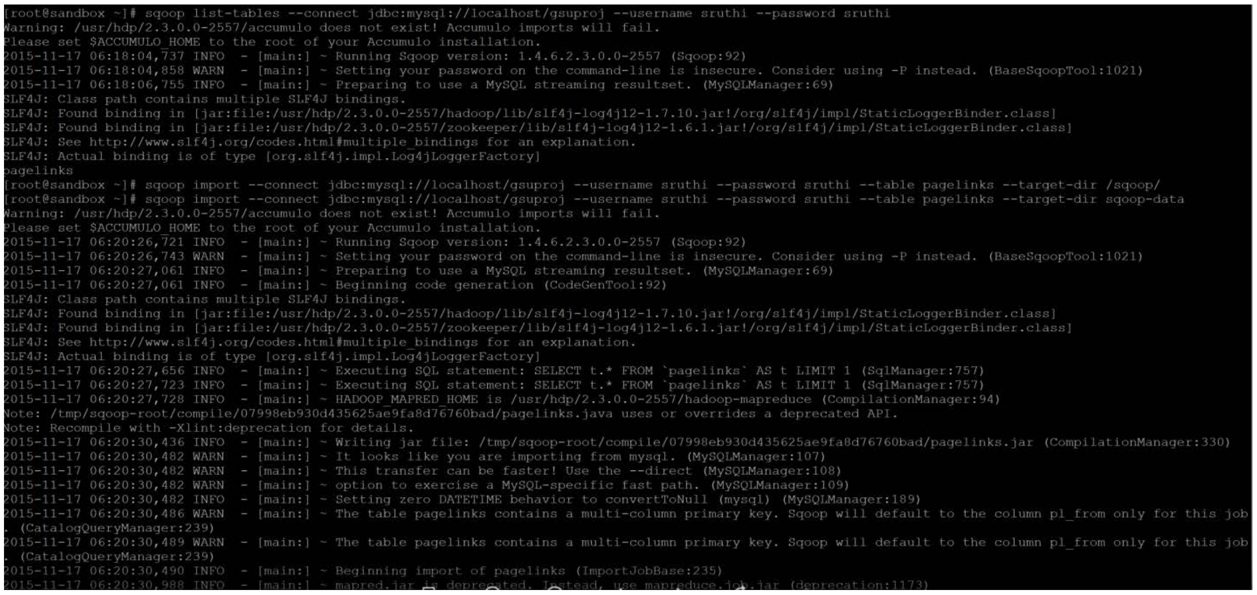
listdatbases in mysql:

sqoop list-databases --connect jdbc:mysql://localhost --username sruthi --password sruthi

list tables in specific database:

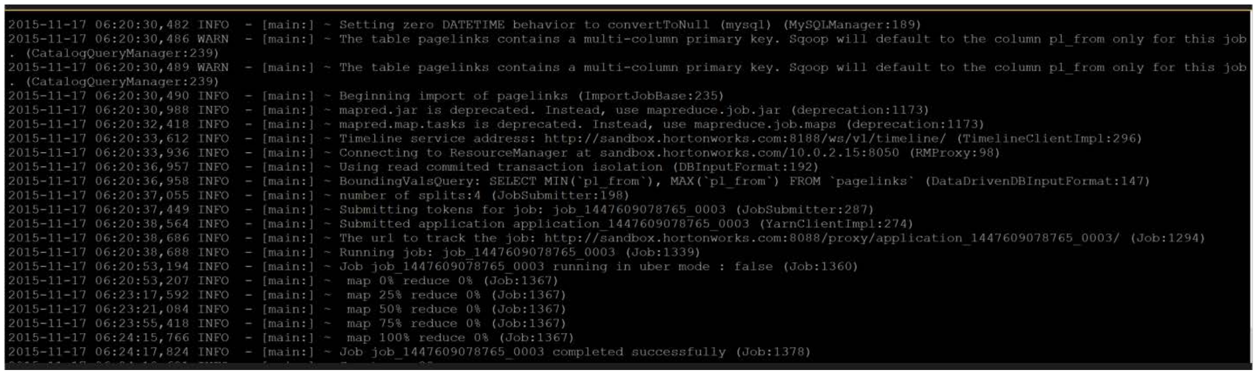
sqoop list-tables --connect jdbc:mysql://localhost/gsuproj --username sruthi --password sruthi

sqoop import --connect jdbc:mysql://localhost/gsuproj --username sruthi --password sruthi --table pagelinks --target-dirsqoop-data



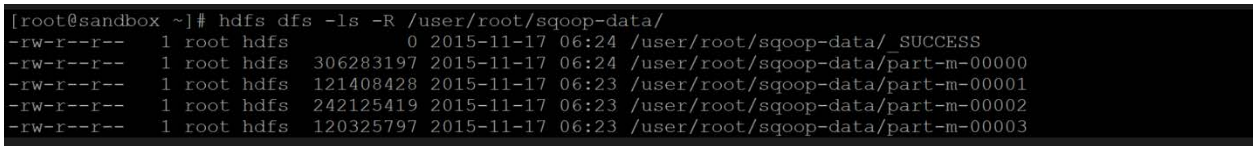
```
[root@sandbox ~]# sqoop list-tables --connect jdbc:mysql://localhost/gsuproj --username sruthi --password sruthi
Warning: /usr/hdp/2.3.0.0-2557/accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
2015-11-17 06:18:04,737 INFO  - [main:] ~ Running Sqoop version: 1.4.6.2.3.0.0-2557 (Sqoop:92)
2015-11-17 06:18:04,858 WARN  - [main:] ~ Setting your password on the command-line is insecure. Consider using -P instead. (BaseSqoopTool:1021)
2015-11-17 06:18:06,755 INFO  - [main:] ~ Preparing to use a MySQL streaming resultset. (MySQLManager:69)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/hdp/2.3.0.0-2557/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/hdp/2.3.0.0-2557/zookeeper/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
pagelinks
[root@sandbox ~]# sqoop import --connect jdbc:mysql://localhost/gsuproj --username sruthi --password sruthi --table pagelinks --target-dir /sqoop/
[root@sandbox ~]# sqoop import --connect jdbc:mysql://localhost/gsuproj --username sruthi --password sruthi --table pagelinks --target-dir sqoop-data
Warning: /usr/hdp/2.3.0.0-2557/accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
2015-11-17 06:20:26,721 INFO  - [main:] ~ Running Sqoop version: 1.4.6.2.3.0.0-2557 (Sqoop:92)
2015-11-17 06:20:26,743 WARN  - [main:] ~ Setting your password on the command-line is insecure. Consider using -P instead. (BaseSqoopTool:1021)
2015-11-17 06:20:27,061 INFO  - [main:] ~ Preparing to use a MySQL streaming resultset. (MySQLManager:69)
2015-11-17 06:20:27,061 INFO  - [main:] ~ Beginning code generation (CodeGenTool:92)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/hdp/2.3.0.0-2557/hadoop/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/hdp/2.3.0.0-2557/zookeeper/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2015-11-17 06:20:27,656 INFO  - [main:] ~ Executing SQL statement: SELECT t.* FROM `pagelinks` AS t LIMIT 1 (SqlManager:757)
2015-11-17 06:20:27,723 INFO  - [main:] ~ Executing SQL statement: SELECT t.* FROM `pagelinks` AS t LIMIT 1 (SqlManager:757)
2015-11-17 06:20:27,728 INFO  - [main:] ~ HADOOP_MAPRED_HOME is /usr/hdp/2.3.0.0-2557/hadoop-mapreduce (CompilationManager:94)
Note: /tmp/sqoop-root/compile/07998eb930d435625ae9fa8d76760bad/pagelinks.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
2015-11-17 06:20:30,436 INFO  - [main:] ~ Writing jar file: /tmp/sqoop-root/compile/07998eb930d435625ae9fa8d76760bad/pagelinks.jar (CompilationManager:330)
2015-11-17 06:20:30,482 WARN  - [main:] ~ It looks like you are importing from mysql. (MySQLManager:107)
2015-11-17 06:20:30,482 WARN  - [main:] ~ This transfer can be faster! Use the --direct (MySQLManager:108)
2015-11-17 06:20:30,482 WARN  - [main:] ~ option to exercise a MySQL-specific fast path. (MySQLManager:109)
2015-11-17 06:20:30,482 INFO  - [main:] ~ Setting zero DATETIME behavior to convertToNull (mysql) (MySQLManager:189)
2015-11-17 06:20:30,486 WARN  - [main:] ~ The table pagelinks contains a multi-column primary key. Sqoop will default to the column pl_from only for this job
. (CatalogQueryManager:239)
2015-11-17 06:20:30,489 WARN  - [main:] ~ The table pagelinks contains a multi-column primary key. Sqoop will default to the column pl_from only for this job
. (CatalogQueryManager:239)
2015-11-17 06:20:30,490 INFO  - [main:] ~ Beginning import of pagelinks (ImportJobBase:235)
2015-11-17 06:20:30,988 INFO  - [main:] ~ mapred.jar is deprecated. Instead, use mapreduce.job.jar (deprecation:1173)
```

sqoop import --connect jdbc:mysql://localhost/gsuproj --username sruthi --password sruthi --table pagelinks
--hive-import



```
2015-11-17 06:20:30,482 INFO  - [main:] ~ Setting zero DATETIME behavior to convertToNull (mysql) (MySQLManager:189)
2015-11-17 06:20:30,486 WARN  - [main:] ~ The table pagelinks contains a multi-column primary key. Sqoop will default to the column pl_from only for this job
. (CatalogQueryManager:239)
2015-11-17 06:20:30,489 WARN  - [main:] ~ The table pagelinks contains a multi-column primary key. Sqoop will default to the column pl_from only for this job
. (CatalogQueryManager:239)
2015-11-17 06:20:30,490 INFO  - [main:] ~ Beginning import of pagelinks (ImportJobBase:235)
2015-11-17 06:20:30,988 INFO  - [main:] ~ mapred.jar is deprecated. Instead, use mapreduce.job.jar (deprecation:1173)
2015-11-17 06:20:32,418 INFO  - [main:] ~ mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps (deprecation:1173)
2015-11-17 06:20:33,612 INFO  - [main:] ~ Timeline service address: http://sandbox.hortonworks.com:8188/ws/v1/timeline/ (TimelineClientImpl:296)
2015-11-17 06:20:33,936 INFO  - [main:] ~ Connecting to ResourceManager at sandbox.hortonworks.com/10.0.2.15:8050 (RMProxy:98)
2015-11-17 06:20:36,957 INFO  - [main:] ~ Using read commited transaction isolation (DBInputFormat:192)
2015-11-17 06:20:36,958 INFO  - [main:] ~ BoundingValsQuery: SELECT MIN(`pl_from`), MAX(`pl_from`) FROM `pagelinks` (DataDrivenDBInputFormat:147)
2015-11-17 06:20:37,055 INFO  - [main:] ~ number of splits:4 (JobSubmitter:198)
2015-11-17 06:20:37,449 INFO  - [main:] ~ Submitting tokens for job: job_1447609078765_0003 (JobSubmitter:287)
2015-11-17 06:20:38,564 INFO  - [main:] ~ Submitted application application_1447609078765_0003 (YarnClientImpl:274)
2015-11-17 06:20:38,686 INFO  - [main:] ~ The url to track the job: http://sandbox.hortonworks.com:8088/proxy/application_1447609078765_0003/ (Job:1294)
2015-11-17 06:20:38,688 INFO  - [main:] ~ Running job: job_1447609078765_0003 (Job:1339)
2015-11-17 06:20:53,194 INFO  - [main:] ~ Job job_1447609078765_0003 running in uber mode : false (Job:1360)
2015-11-17 06:20:53,207 INFO  - [main:] ~  map 0% reduce 0% (Job:1367)
2015-11-17 06:23:17,592 INFO  - [main:] ~  map 25% reduce 0% (Job:1367)
2015-11-17 06:23:21,084 INFO  - [main:] ~  map 50% reduce 0% (Job:1367)
2015-11-17 06:23:55,418 INFO  - [main:] ~  map 75% reduce 0% (Job:1367)
2015-11-17 06:24:15,766 INFO  - [main:] ~  map 100% reduce 0% (Job:1367)
2015-11-17 06:24:17,824 INFO  - [main:] ~ Job job_1447609078765_0003 completed successfully (Job:1378)
```

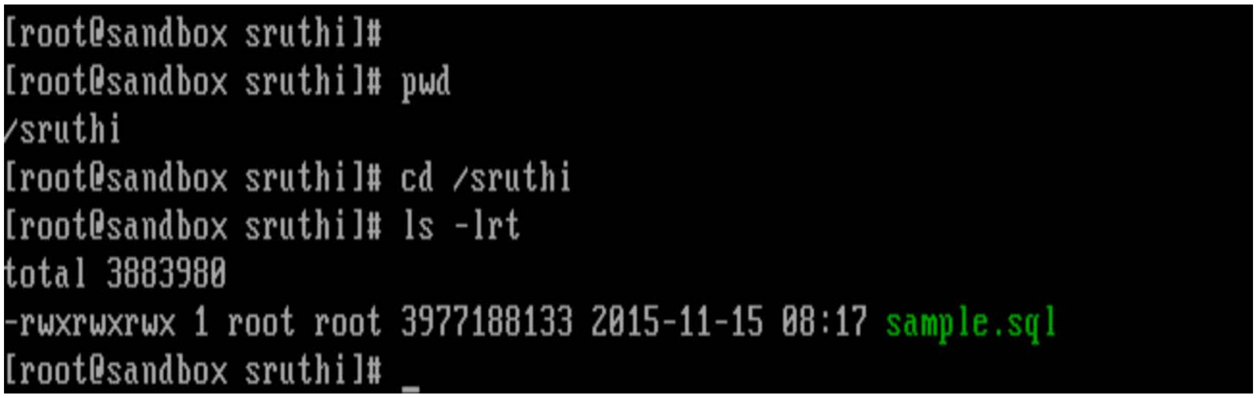
Converting data in to file and can see the file names:



```
[root@sandbox ~]# hdfs dfs -ls -R /user/root/sqoop-data/
-rw-r--r--   1 root hdfs          0 2015-11-17 06:24 /user/root/sqoop-data/_SUCCESS
-rw-r--r--   1 root hdfs  306283197 2015-11-17 06:24 /user/root/sqoop-data/part-m-00000
-rw-r--r--   1 root hdfs  121408428 2015-11-17 06:23 /user/root/sqoop-data/part-m-00001
-rw-r--r--   1 root hdfs  242125419 2015-11-17 06:23 /user/root/sqoop-data/part-m-00002
-rw-r--r--   1 root hdfs  120325797 2015-11-17 06:23 /user/root/sqoop-data/part-m-00003
```

Step 2: Store file into Hadoop cluster

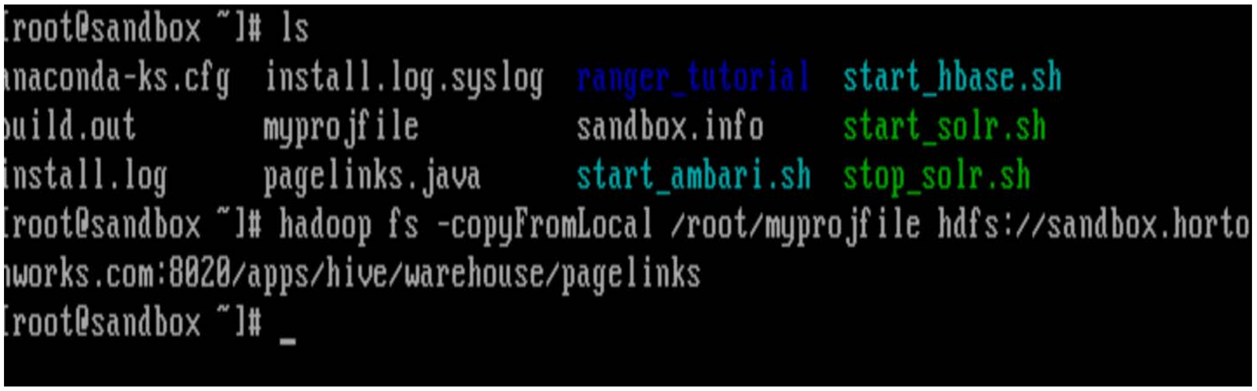
Connect to the server



```
[root@sandbox sruthi]#
[root@sandbox sruthi]# pwd
/sruthi
[root@sandbox sruthi]# cd /sruthi
[root@sandbox sruthi]# ls -lrt
total 3883980
-rwxrwxrwx 1 root root 3977188133 2015-11-15 08:17 sample.sql
[root@sandbox sruthi]# _
```

Move file into HDFS by using below command

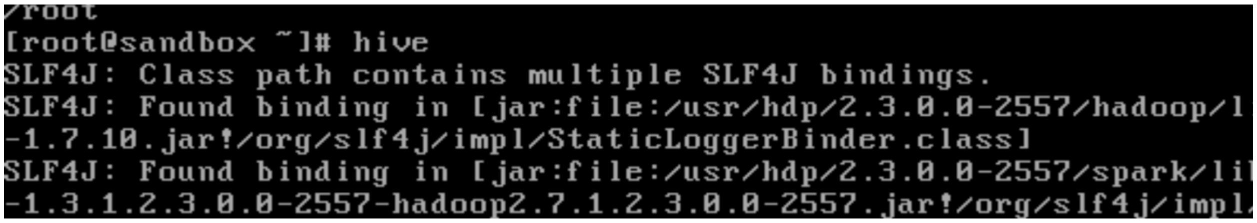
hadoopfs -copyFromLocal/root/myprojfile
hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse/pagelinks



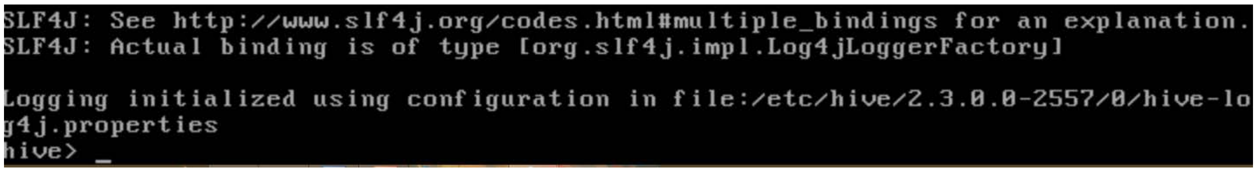
```
root@sandbox ~]# ls
naconda-ks.cfg  install.log.syslog  ranger_tutorial  start_hbase.sh
uild.out        myprojfile          sandbox.info     start_solr.sh
nstall.log      pagelinks.java      start_ambari.sh  stop_solr.sh
root@sandbox ~]# hadoop fs -copyFromLocal /root/myprojfile hdfs://sandbox.horto
nworks.com:8020/apps/hive/warehouse/pagelinks
root@sandbox ~]# _
```

Step 3: Read Data from HIVE

Connect to the Hive from root



```
/root
[root@sandbox ~]# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/hdp/2.3.0.0-2557/hadoop/l
-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/hdp/2.3.0.0-2557/spark/li
-1.3.1.2.3.0.0-2557-hadoop2.7.1.2.3.0.0-2557.jar!/org/slf4j/impl
```



```
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

Logging initialized using configuration in file:/etc/hive/2.3.0.0-2557/0/hive-lo
g4j.properties
hive> _
```

Create table command:

CREATE external TABLE pagelinks (

pl_from string,

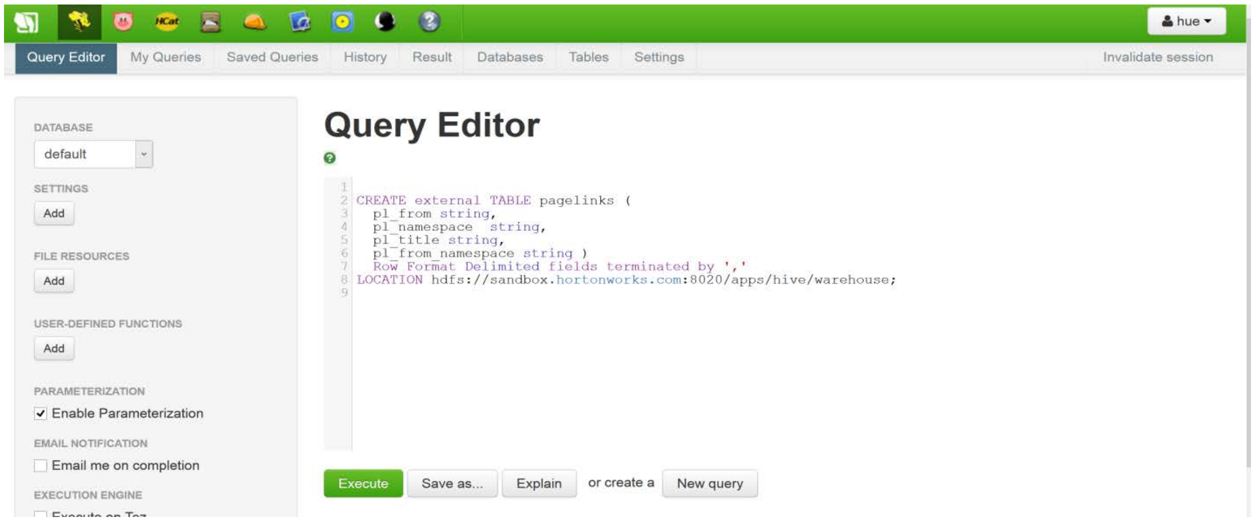
pl_namespace string,

pl_title string,

pl_from_namespace string)

  Row Format Delimited fields terminated by '~'

LOCATION hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse;



```

CREATE external TABLE pagelinks (
  pl_from string,
  pl_namespace  string,
  pl_title string,
  pl_from_namespace string )
  Row Format Delimited fields terminated by ','
LOCATION hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse;

```

Enable Parameterization [✓]

LOAD DATA INPATH hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse
INTO TABLE pagelinks

5 ProjectInternal/external Interface Impactsand Specification

Hadoop has become one of the most important technologies, the key factors of hadoop are as follows:

Low Cost :

Hadoop is an free open Source frame work, and uses commodity hardware to store substantial amount of data. Hadoop additionally offers a practical stockpiling answer for organizations' blasting information sets. The issue with customary social database administration frameworks is that it is amazingly taken a toll restrictive to scale to such an extent so as to process such huge volumes of information. With an end goal to lessen costs, numerous organizations in the past would have needed to down-example information and characterize it in view of specific presumptions as to which information was the most significant. The crude information would be erased, as it would be excessively taken a toll restrictive, making it impossible to keep. While this methodology may have worked in the short term, this implied that when business needs changed, the complete crude information set was not accessible, as it was so costly it was not possible store. Hadoop, then again, is outlined as a scale-out structural planning that can reasonably store the majority of an organization's information for later utilize. The expense reserve funds are stunning: as opposed to costing thousands to a huge number of pounds every terabyte, Hadoop offers figuring and stockpiling capacities for many pounds every terabyte.

Flexible:

Hadoop empowers organizations to effectively get to new information sources and tap into distinctive sorts of information (both organized and unstructured) to create esteem from that information. This implies organizations can utilize Hadoop to get profitable business bits of knowledge from information sources, for example, social networking, email discussions information. Moreover, Hadoop can be utilized for a wide mixed bag of purposes, for example, log handling, suggestion frameworks,

information warehousing, business sector battle investigation and misrepresentation discovery.

Computing power:

Hadoop is a distributed processing model, so it can prepare vast measure of information. The additionally registering hubs you utilize.

Fast:

Hadoop's unique storage method is based on a distributed file system that basically 'maps' data wherever it is located on a cluster. The tools for data processing are often on the same servers where the data is located, resulting in much faster data processing. If you're dealing with large volumes of unstructured data, Hadoop is able to efficiently process terabytes of data in just minutes petabytes in hours.

Storage Flexibility:

Unlike traditional relational databases, you don't have to preprocess data before storing it. And that includes unstructured data like text, images and videos. You can store as much data as you want and

decide how to use it later.

Inherent data protection and self-healing capabilities:

Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. And it automatically stores multiple copies of all data.

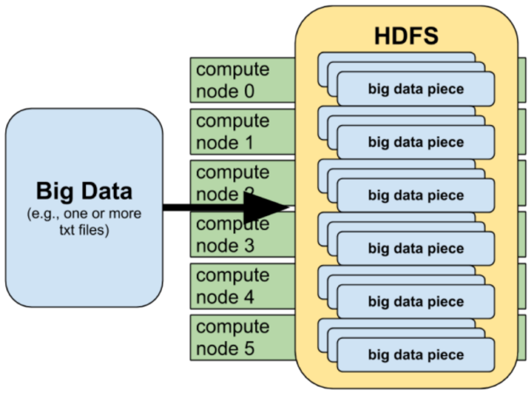
6 *Functional overview*

The idea underpinning map-reduce–bringing compute to the data instead of the opposite–should sound like a very simple solution to the I/O bottleneck inherent in traditional parallelism. However, the devil is in the details, and implementing a framework where a single large file is transparently diced up and distributed across multiple physical computing elements (all while appearing to remain a single file to the user) is not trivial.

Hadoop, perhaps the most widely used map-reduce framework, accomplishes this feat using HDFS, the Hadoop Distributed File System. HDFS is fundamental to Hadoop because it provides the data chunking and distribution across compute elements necessary for map-reduce applications to be efficient. Since we're now talking about an actual map-reduce implementation and not an abstract concept, let's refer to the abstract compute elements now as compute nodes.

HDFS exists as a filesystem into which you can copy files to and from in a manner not unlike any other filesystem. Many of the typical commands for manipulating files (ls, mkdir, rm, mv, cp, cat,tail, and chmod, to name a few) behave as you might expect in any other standard filesystem (e.g., Linux's ext4).

The magical part of HDFS is what is going on just underneath the surface. Although it appears to be a filesystem that contains files like any other, in reality those files are distributed across multiple physical compute nodes:



When you copy a file into HDFS as depicted above, that file is transparently sliced into 64 MB "chunks" and replicated three times for reliability. Each of these chunks are distributed to various compute nodes in the Hadoop cluster so that a given 64 MB chunk exists on three independent nodes. Although physically chunked up and distributed in triplicate, all of your interactions with the file on HDFS still make it appear as the same single file you copied into HDFS initially. Thus, HDFS handles all of the burden of slicing, distributing, and recombining your data for you.

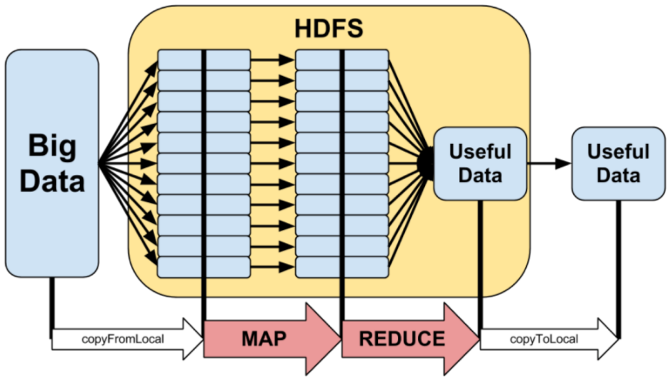
HDFS's chunk size and replication The 64 MB chunk (block) size and the choice to replicate your data three times are only HDFS's default values. These decisions can be changed:

- The 64 MB block size can be modified by changing the dfs.block.size property in hdfs-site.xml. It is common to increase this to 128 MB in production environments.
- The replication factor can be modified by changing the dfs.replication property in hdfs-site.xml. It can also be changed on a per-file basis by specifying -D dfs.replication=1on your -put command line, or using the hadoopdfs -setrep -w 1 command.

Map-Reduce Jobs

HDFS is an interesting technology in that it provides data distribution, replication, and automatic recovery in a user-space filesystem that is relatively easy to configure and, conceptually, easy to understand. However, its true utility comes to light when map-reduce jobs are executed on data stored in HDFS.

As the name implies, map-reduce jobs are principally comprised of two steps: the map step and the reduce step. The overall workflow generally looks something like this:



The left half of the diagram depicts the HDFS magic described in the previous section, where thehadoopdfs -copyFromLocal command is used to move a large data file into HDFS and it is automatically replicated and distributed across multiple physical compute nodes. While this step of moving data into HDFS is not strictly a part of a map-reduce job (i.e., your dataset may already have a permanent home on HDFS just like it would any other filesystem), a map-reduce job's input data must already exist on HDFS before the job can be started.

The Map Step

Once a map-reduce job is initiated, the map step

1. Launches a number of parallel mappers across the compute nodes that contain chunks of your input data
2. For each chunk, a mapper then "splits" the data into individual lines of text on newline characters (\n)
3. Each split (line of text that was terminated by \n) is given to your mapper function
4. Your mapper function is expected to turn each line into zero or more key-value pairs and then "emit" these key-value pairs for the subsequent reduce step

That is, the map step's job is to transform your raw input data into a series of key-value pairs with the expectation that these parsed key-value pairs can be analyzed meaningfully by the reduce step. It's perfectly fine for duplicate keys to be emitted by mappers.

The decision to split your input data along newline characters is just the default behavior, which assumes your input data is just an ascii text file. You can change how input data is split before being passed to your mapper function using alternate InputFormats.

The Reduce Step

Once all of the mappers have finished digesting the input data and have emitted all of their key-value pairs, those key-value pairs are sorted according to their keys and then passed on to the reducers. The reducers are given key-value pairs in such a way that all key-value pairs sharing the same key always go to the same reducer. The corollary is then that if one particular reducer has one specific key, it is guaranteed to have all other key-value pairs sharing that same key, and all those common keys will be in a continuous strip of key-value pairs that reducer received.

Your job's reducer function then does some sort of calculation based on all of the values that share a common key. For example, the reducer might calculate the sum of all values for each key (e.g., the word count example). The reducers then emit key-value pairs back to HDFS where each key is unique, and each of these unique keys' values are the result of the reducer function's calculation.

6.1 Impact

VMware's new vSphere Big Data Extensions (BDE) commercializes its open source Project Serengetito make it dead easy for enterprise admins to spin and up down virtual Hadoop clusters at will. Now that VMware has made it clear that Hadoop is going to be fully supported as a virtualized workload in enterprise vSphere environments, here at Taneja Group we expect a rapid pickup in Hadoop adoption across organizations of all sizes
Big Data, Virtually
However, Hadoop is all about mapping parallel compute jobs intelligently over massive amounts of distributed data. Cluster deployment and operation are becoming very easy for the virtual admin. But in a virtual environment where storage can be effectively abstracted from compute clients, there are some important complexities and opportunities to consider when designing the underlying storage architecture. Some specific concerns with running Hadoop in a virtual environment include considering how to configure virtual data nodes, how to best utilize local hypervisor server DAS, and when to think about leveraging external SAN/NAS.

Big Data, Virtually
The main idea behind virtualizing Hadoop is to take advantage of deploying Hadoop scale-out nodes as virtual machines instead of as racked commodity physical servers. Clusters can be provisioned on-demand and elastically expanded or shrunk. Multiple Hadoop virtual nodes can be hosted on each hypervisor physical server, and as virtual machines can be easily allocated more or less resource for a given application. Hypervisor level HA/FT capabilities can be brought to bear on production Hadoop apps. VMware's BDE even includes QoS algorithms that help prioritize clusters dynamically, shrinking lower-priority cluster sizes as necessary to ensure high-priority cluster service.

Obviously, one of the big concerns with virtualizing Hadoop is about performance. Much of Hadoop's value lies in how it effectively executes parallel algorithms over distributed data chunks. Hadoop takes advantage of high data "locality" by spreading out big data over many nodes using HDFS (Hadoop Distributed File System). It then farms out parallelized compute tasks local to each data node for initial processing (the "map" part of MapReduce implemented by job and task trackers).

The design, with each scale-out physical node hosting both local compute and a share of data, is intended to support applications like searching and scoring. These applications might often need to crawl through all the data of massively large data sets, which are commonly made up of low-level semi- or unstructured text and documents.

Commonly, each HDFS data node will be assigned raw physical host server DAS disks directly by the hypervisor. HDFS will then replicate data across data nodes, by default making two copies on different data nodes. On a physical cluster, replicates are placed on different server nodes by definition (one data node per server). HDFS also knows to place the second replicate on a different "rack" of nodes to help avoid rack level loss.

In the virtual world, Hadoop must become aware of the hypervisor grouping of virtual nodes in order to ensure good physical data placement and subsequent job/task assignment. This virtual awareness is implemented by the Hadoop Virtual Extensions (HVE) that VMware contributed into Apache Hadoop 1.2.

Hadoop Virtual Extensions
The Hadoop Virtual Extensions do break the virtual abstraction between application and physical hosting. But in some ways, the Hadoop platform can be seen as another layer of the virtualization, adding scale-out data and computing management to the hypervisor.

The HVE essentially inserts a new level of “node group” into the Hadoop hierarchy between nodes and racks. Node groups represent the set of virtual Hadoop nodes on each given hypervisor server to help inform Hadoop and HDFS management algorithms.

The effect is that Hadoop can maintain knowledge of “data locality” even in the virtual environment to keep compute tasks close to required data for performance, and ensure optimal placement of replicates for fault tolerance.

Data Node Options
When you virtualize Hadoop nodes, you also have the option to separate the compute side (task trackers, et.al.) from the data node and place them each in different virtual machines. If the compute node and data node virtual machines still reside on the same hypervisor server, then they can effectively communicate over a virtual network “in-memory” and won’t suffer any significant physical network latencies. HVE can ensure that this data local relationship is maintained for performance.

Separating out the data node from compute nodes gives you orthogonal scaling and the option to host multiple compute nodes sharing a single data node. This new flexibility enables optimizing the utilization of the host physical server resources, although getting the ratios right for each application might require a lot of experimentation.
In fact, HDFS can be offered as a service itself, managed as a more permanent data repository, while various compute “applications” can come and go quite dynamically. In this way, HDFS can now serve as a scale out virtual storage appliance.

Big Data SAN?
One of the cost-compelling reasons to look at a physical Hadoop architecture is to avoid expensive SANs, especially as the data sets grow larger. Yet in the virtual environment it may make sense to consider SAN storage for some big data sets.
One reason is that provisioning compute-only virtual Hadoop clusters is quite simple with VMware’s BDE GUI, but throwing around big data sets is still going to be a challenge. By hosting the data on external shared storage, provisioning virtual Hadoop hosting becomes almost trivial. And hypervisor features like DRS and HA can be fully leveraged. At EMC World 2013, Pat Gelsinger readily demonstrated spinning up and down virtual Hadoop clusters using external Isilon storage.

Another reason to look at SAN storage is if you have data governance concerns. HDFS is not easy to backup, protect, secure or audit. SANs of course, are built with great data protection (and use fewer disks for RAID than triplicate replication) and snapshots. It’s easy to imagine some big data applications where the data is critical enough to want to protect and rollback if necessary. With an eye towards ensuring some high performance networking, performance from SANs can of course provide more throughput than server DAS. It is worth mentioning disk failure recovery here too, because with big data on lots of disks, failures become quite common.
In a normal Hadoop cluster, a local disk failure shuts down that node, and Hadoop then works around it. In a virtual environment, a disk failure might shut down the data node, but multiple virtual data nodes can be configured per hypervisor server. And a disk failure that sidelines a virtual data node will not take down any other virtual Hadoop nodes on that hypervisor. With SAN storage, a highly available Hadoop application might never know that disk failures have even happened.
There are a number of reasons why virtualizing Hadoop makes sense in many usage scenarios. As a virtual workload, Hadoop can achieve comparable performance to physical hosting in a broad set of expected usage scenarios while further helping consolidate and optimize IT infrastructure investments.

At this point, thousand node clusters with multiple PBs of data in continuous use aren’t likely virtualization candidates. But we think that most organizations have some big data opportunities in the 10-20TB range, and they could be extracting value from that data if only their IT shops could offer scale-out analytical solutions as a cost-effective service.

With a virtual Hadoop capability, a single big data set can be readily shared “in-place” between multiple virtualized Hadoop clusters. That creates an opportunity to serve multiple clients with the same storage. By eliminating multiple copies of big data sets, reducing the amount of data migration, and ensuring higher availability and data protection, Hadoop becomes more manageable and readily supported as an enterprise production application.

7 Open Issues

Hadoop tracks both bugs and enhancement requests using JIRA. Input before filing a request search the Apache JIRA database. Check the users mailing lists, both by searching the archives and by asking questions. Common Hadoop Common issues are tracked in the HADOOP Jira instance. HDFS issues are tracked in the HDFS Jira instance. YARN issues are tracked in the YARN Jira instance. MapReduce issues are tracked in the MAPREDUCE Jira instance.

HDFS is the primary distributed storage used by Hadoop applications. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data. The HDFS Architecture Guide describes HDFS in detail. This user guide primarily deals with the interaction of users and administrators with HDFS clusters. The HDFS architecture diagram depicts basic interactions among NameNode, the DataNodes, and the clients. Clients contact NameNode for file metadata or file modifications and perform actual file I/O directly with the DataNodes.

The following are some of the salient features that could be of interest to many users.

- Hadoop, including HDFS, is well suited for distributed storage and distributed processing using commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. MapReduce, well known for its simplicity and applicability for large set of distributed applications, is an integral part of Hadoop.
- HDFS is highly configurable with a default configuration well suited for many installations. Most of the time, configuration needs to be tuned only for very large clusters.
- Hadoop is written in Java and is supported on all major platforms.
- Hadoop supports shell-like commands to interact with HDFS directly.
- The NameNode and Datanodes have built in web servers that makes it easy to check current status of the cluster.

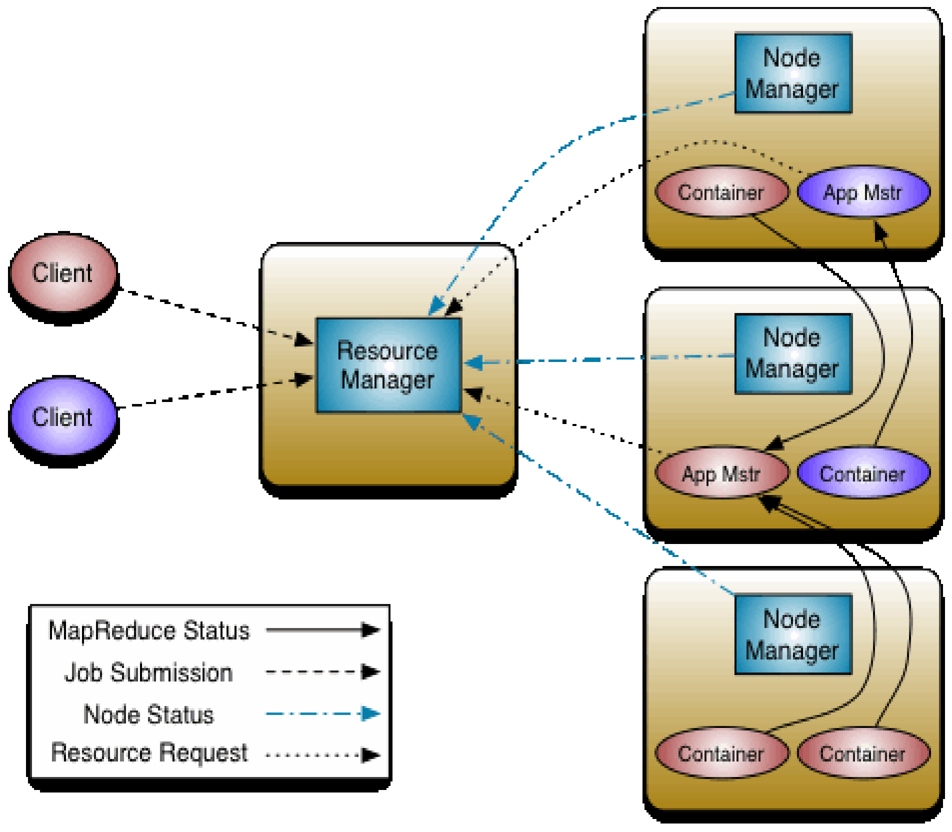
page. It lists the DataNodes in the cluster and basic statistics of the cluster. The web interface can also be used to browse the file system . Hadoop includes various shell-like commands that directly interact with HDFS and other file systems that Hadoop supports. The command bin/hdfsdfs help lists the commands supported by Hadoop shell. Furthermore, the command bin/hdfsdfsn name displays more detailed help for a command. These commands support most of the normal files system operations like copying files, changing file permissions, etc. It also supports a few HDFS specific operations like changing replication of files. The bin/hdfsdfsadmin command supports a few HDFS administration related operations. This help command lists all the commands currently supported. Some samples are below.

When Hadoop is upgraded on an existing cluster, as with any software upgrade, it is possible there are new bugs or incompatible changes that affect existing applications and were not discovered earlier. In any non-trivial HDFS installation, it is not an option to loose any data, let alone to restart HDFS from scratch. HDFS allows administrators to go back to earlier version of Hadoop and rollback the cluster to the state it was in before the upgrade. HDFS upgrade is described in more detail in Hadoop Upgrade Wiki page. HDFS can have one such backup at a time. Before upgrading, administrators need to remove existing backup using bin/hadoopdfsadmin -finalizeUpgrade command. The following briefly describes the typical upgrade procedure:

- Before upgrading Hadoop software, finalize if there an existing backup. dfsadmin -upgradeProgress status can tell if the cluster needs to be finalized.
- Stop the cluster and distribute new version of Hadoop.
- Run the new version with -upgrade option (bin/start-dfs.sh -upgrade).
- Most of the time, cluster works just fine. Once the new HDFS is considered working well (may be after a few days of operation), finalize the upgrade. Note that until the cluster is finalized, deleting the files that existed before the upgrade does not free up real disk space on the DataNodes.

The file permissions are designed to be similar to file permissions on other familiar platforms like Linux. Currently, security is limited to simple file permissions. The user that starts NameNode is treated as the superuser for HDFS. Future versions of HDFS will support network authentication protocols like Kerberos for user authentication and encryption of data transfers. The details are discussed in the Permissions Guide. Hadoop currently runs on clusters with thousands of nodes. lists some of the organizations that deploy Hadoop on large clusters. HDFS has one NameNode for each cluster. Currently the total memory available on NameNode is the primary scalability limitation. On very large clusters, increasing average size of files stored in HDFS helps with increasing cluster size without increasing memory requirements on NameNode. The default configuration may not suite very large clusters.

Yarn MapReduce has undergone a complete overhaul in hadoop-0.23 and we now have, what we call, MapReduce 2.0 (MRv2) or YARN.The fundamental idea of MRv2 is to split up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM). An application is either a single job in the classical sense of Map-Reduce jobs or a DAG of jobs.The ResourceManager and per-node slave, the NodeManager (NM), form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.



Map Reduce Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. The MapReduce framework consists of a single master ResourceManager, one slave NodeManager per cluster-node, and MRAppMaster per application (see YARN Architecture Guide). Minimally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the ResourceManager which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client. Although the Hadoop framework is implemented in Java™, MapReduce applications need not be written in Java. Mapper maps input key/value pairs to a set of intermediate key/value pairs.

Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job.Overall, Mapper implementations are passed the Job for the job via the Job.setMapperClass(Class) method. The framework then calls map(WritableComparable, Writable, Context) for each key/value pair in the InputSplit for that task. Applications can then override the cleanup(Context) method to perform any required cleanup. Output pairs do not need to be of the same types as input pairs. A given input pair may map to zero or many output pairs. Output pairs are collected with calls to context.write(WritableComparable, Writable). Applications can use the Counter to report its statistics. All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the Reducer(s) to determine the final output. Users can control the grouping by specifying a Comparator via Job.setGroupingComparatorClass(Class). The Mapper outputs are sorted and then partitioned per Reducer. The total number of partitions is the same as the number of reduce tasks for the job. Users can control which keys (and hence records) go to which Reducer by implementing a custom Partitioner. Users can optionally specify a combiner, via Job.setCombinerClass(Class), to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer. The intermediate, sorted outputs are always stored in a simple (key-len, key, value-len, value) format. Applications can control if, and how, the intermediate outputs are to be compressed and the CompressionCodec to be used via the Configuration.

8 References

1. http://hadoop.apache.org/
2. http://www.cloudera.com/content/cloudera/en/about/hadoop-and-big-data.html
3. http://hortonworks.com/hadoop
4. http://hadoop.apache.org/
5. http://www.cloudera.com/content/cloudera/en/about/hadoop-and-big-data.html
6. http://www.cloudera.com/content/cloudera/en/about/hadoop-and-big-data.html
7. http://www-01.ibm.com/software/data/infosphere/hadoop
8. http://www.usa.gov/About/developer-resources/1usagov.shtml
9. http://www.enterprisestorageforum.com/storage-management/the-impact-of-virtualized-hadoop.html
10. http://searchcio.techtarget.com/opinion/Hadoop-20s-deep-impact-on-big-data-and-big-data-technologies
11. http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-hs/dependency-analysis.html
12. http://www.bodhtree.com/blog/2012/09/08/what-is-hive-it%E2%80%99s-interaction-with-hadoop-and-big-data/
13. https://www.quora.com/How-does-the-rising-popularity-of-Hadoop-impact-legacy-data-warehouse-vendors-like-Teradata
14. http://conferences.oreilly.com/strata/strataeu2014/public/schedule/detail/39636
15. http://www.cloudera.com/content/www/en-us/campaign/data-impact-awards.html