Survey of Apache Big Data Stack

Supun Kamburugamuve
For the PhD Qualifying Exam
12/16/2013

Advisory Committee
Prof. Geoffrey Fox
Prof. David Leake
Prof. Judy Qiu

Table of Contents

1. Introduction

Over the last decade there has being an explosion of data. Large amounts of data are generated from scientific experiments, government records, and large Internet sites and sensors networks. The term Big Data was introduced to identify such data that cannot be captured, curated, managed or processed by traditional tools in a reasonable amount of time. With the amount of data generated increasing everyday without an end in sight, there is a strong need for storing, processing and analyzing this huge volumes of data. A rich set of tools ranging from storage management to data processing and to data analytics has being developed to this specific purpose. Because of the shear volume of data, a large amount of computing power and space is required to process this data. Having specialized hardware like Super Computing infrastructures for doing such processing is not economically feasible most of the time. Large clusters of commodity hardware are a good economical alternative for getting the required computing power. But such large clusters of commodity hardware impose challenges that are not seen in traditional high-end hardware and big data frameworks must be specifically designed in order to overcome these challenges to be successful.

Open source software is playing a key role in the big data field and there is a growing trend to make the big data solutions open and free to the general public. The open source software is dominating the big data solutions space and we hardly hear the names of proprietary software giants like Microsoft, Oracle and IBM in this space. Open source software development is rapidly changing the innovation in the big data field and how we generally think about big data solutions.

The open source software movement officially began in 1983 with the start of the GNU Project by Richard Stallman. Open source software development is well studied in the Information Science research and the development methods used by the open source communities have proven to be successful in projects requiring the human innovation and passion. One of the most important factors for a successful open source project is the community around the project. The community drives the innovation and development of an open source project. Well functioning diverse communities can create software solutions that are robust and facilitating diverse set of requirements.

Apache Software Foundation (ASF) [1] is a non-profit open source software foundation, which has put itself in a very important position in the big data space. The software foundation has a diverse development and user community spreading the globe and is home to some of the most widely used software projects. The community is considered important above all other requirements in ASF and it has being the main reason behind its success. As a result, ASF provides a flexible and agile environment for the open source project development along with infrastructure and legal framework necessary for maintaining successful projects. Because of its success as an open source foundation, ASF has attracted a large chunk of successful big data projects in the past few years.

The other leading open source software platform for big data projects is GitHub. GitHub is a git based code repository for open source projects and it doesn't provide the legal and organizational framework provided by ASF. GitHub is an ad-hoc platform for developing open source software and communities are formed in an add-hoc manner around the successful projects. Software foundations like ASF, Universities and companies like Netflix, LinkedIn use GitHub for hosting their open source projects. There are successful big data projects developed at GitHub and communities are formed around these products.

There is a recent trend in large Internet companies to make most of their entire operational codes available as open source free platforms. Netflix [2] and LinkedIn [3] are pioneers in making their source code open to the public. Various projects created by large software companies like Yahoo, Facebook and Twitter is being donated to the public through the open source software foundations. The process is mutually benefiting both the community and the original software creators. The original software creators get their code exposed to a diverse community and this helps the products to mature fast and evolve quickly. The open source software gets battle tested in all kinds of scenarios for free, which makes the product resilient. One of the most rewarding things about making software open source may be the gaining of high credibility and trust among the peer developers for the leaders of open source software projects.

Even though there are many open source platforms for big data projects, Apache Software Foundation has immerged as the clear leader in the open source Big Data space, hosting the key big data projects. ASF has a complete stack of big data solutions ranging from the basic storage management to complex data analytics. These projects are mostly autonomous projects intertwined by the technologies they use but they somehow form a complete solution stack for

big data. It is interesting to know the inter-relationships among these projects and the internal of these projects. In this paper we will analyze the big data echo system in ASF by categorizing the projects in to a layered big data stack.

The first part of the report will focus on the overall layered architecture of the big data stack and rest of the report will discuss each layer starting from the bottom.

2. The ASF Big Data Stack

The open source big data movement in Apache began with the introduction of Hadoop to the foundation in 2007. The initial version of Hadoop was developed by Yahoo and denoted to ASF. The Hadoop project was originally developed as a large-scale computational infrastructure for the Yahoo search engine. The project is based on the Google MapReduce framework [4] and Google File System [5]. The original designers of the Hadoop had the problem of processing large volume of data using commodity hardware. The first problem was, how to access a large volume of data in a distributed environment. Having this capability extracted out as a separate layer makes the development of the computation layer robust. Hadoop Distributed File system (HDFS) provides the capability to store large volume of data in commodity hardware in large clusters. Also it provides the APIs necessary to read this data efficiently. HDFS can be seen as a trim down version of a distributed file system with emphasis on storing large data and reading them frequently. The computation framework built on top of the file systems is a MapReduce based computation framework for data parallel applications. Because of the success of Hadoop, large number of big data projects was built on and around Hadoop creating a complete big data project stack in Apache. With the Hadoop version 2 the architecture of Hadoop has changed significantly to support such a wide variety of projects around it. The new generation of Hadoop is called Apache Yarn and it adds a generic resource management layer to provide better resource utilization in clusters for big data projects.

We have identified five layers in the ASF big data projects. The layered architecture is shown in the figure 1. The bottom layer is the distributed file storage and this serves as the basic infrastructure for all the other layers. Next to the file storage layer there is a cluster resource management layer focusing on managing large clusters of commodity hardware resources and provide a platform for executing the upper layer products in a resource efficient manner. Next we have large-scale structured storage solutions and the resource management layer can manage these and they can use the distributed file storage for data storage. Then we have the large-scale data processing frameworks. These frameworks can process data stored in distributed file systems, structured storages or data coming from the network. High level abstractions has being built on top of these processing layers to make the data processing more intuitive and easy at the next layer. In the top layer there are frameworks for analyzing the data using all of the below layers. The various tools developed to facilitate the big data projects are included in to a separate layer orthogonal to all the other layers.

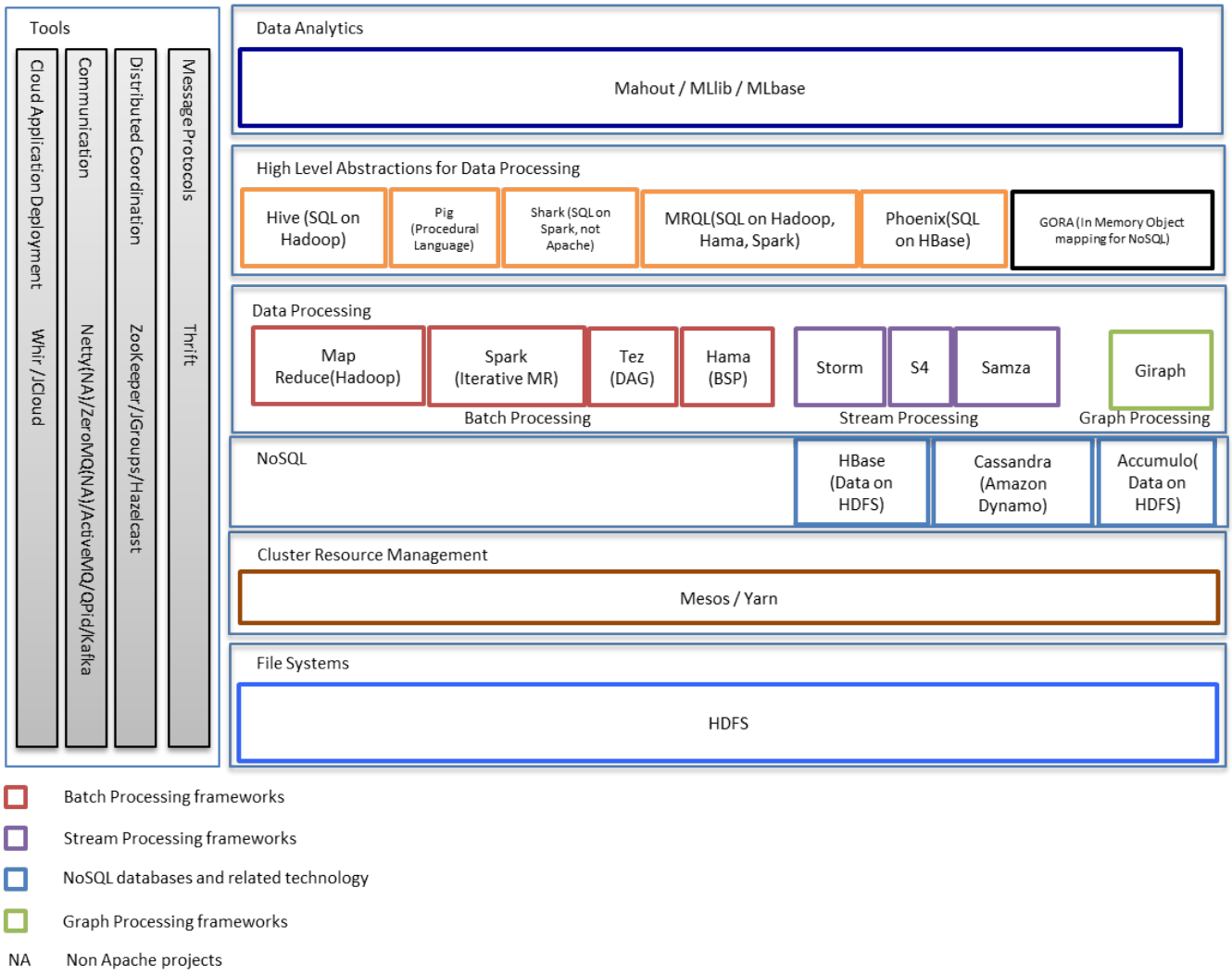

Figure 1 Apache Big Data Stack

2.1 The Tools Layer

Large number of tools has developed around the big data projects. Some of the projects are Apache projects and some are from other open source platforms. In this section we will focus on Apache Thrift and Apache ZooKeeper as they are widely used across the big data products.

2.1.1 Apache Thrift

Apache Thrift is a cross language service development framework. It is used as a scalable remote procedure call framework to facilitate the communication among heterogeneous software components of large software projects. Thrift has an interface definition language (IDL) along with a code generation for most of the widely used programming languages making it possible to connect virtually any two software components. Thrift has code generation support for languages like C++, C#, Cocoa, D, Delphi, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, Smalltalk and more other. The Thrift IDL consists of the definitions of types and services. A type can be a primitive type or a composite type (struct). A service contains method definitions. The Thrift IDL has support for exceptions, service inheritance and sparse structs.

Thrift uses an efficient and compact binary message format for serializing the object in to the wire. Thrift is used by the big data projects to define interfaces for the client applications. A client application can generate language bindings for the Thrift service interfaces and communicate using highly efficient binary messages. Thrift has

decoupled the wire transport from the message serialization. A user can use variety of wire transports to communicate between a Thrift service and a Thrift client. Also a new transport can be plugged in to Thrift for situation where there are more fine-grained requirements for the transport.

2.1.2 Apache ZooKeeper

Apache ZooKeeper [6] is an Internet scale distributed coordination system developed at Yahoo and donated to Apache Software Foundation. ZooKeeper provides a centralized service for coordinating distributed applications. ZooKeeper provides few consistency guarantees upon which client applications can be built complex coordinating protocols.

Consistency Guarantees

ZooKeeper provides the following primitive guarantees to the client applications.

1. Sequential Consistency - Client updates are applied on the order they were sent
2. Atomicity - Updates are applied as a whole without any partial results
3. Single System - Client will see the distributed ZooKeeper system as a single system
4. Reliability - Once an update has made, it is persisted
    - The ZooKeeper guarantees that the update is applied and persisted if the client gets a response
    - If an update is seen by a client, that update will not get rolled back in the event of a recovery
5. Timeliness - The client view of the system is guaranteed to be up to date within a certain time bound

With these guarantees, an application can use the ZooKeeper client to build higher-level coordination functions such as leader election, barriers, queues, and read/write revocable locks.

Data Model

ZooKeeper service stores information in ZNodes. ZNodes are arranged in a tree structure that closely resembles a Unix file system. The difference between a ZNode and a file is that, a ZNode can contain data as well as child nodes unlike a regular Unix file. A ZNode is referenced by a path, which is always expressed as canonical, absolute, slash-separated path similar to an absolute path in Unix file system. Ephemeral node is a special node that only exists until the session that created the node is active. To avoid heavy polling by clients on nodes, ZooKeeper has a concept of watches. A client can set a watch on a ZNode to listen for changes.

2.1.3 Apache Curator

Apache Curator is a library built on top of ZooKeeper client API to provide high-level coordination services on top of the low level APIs provided by the ZooKeeper. It provides API methods to create distributed locks, leader election without burdening the user with details of the distributed coordination. Also Curator has provided libraries for robust client connection management and various utilities for managing ZooKeeper objects.

2.2 File Systems

The lowest layer of the big data is the immensely successful Hadoop file system (HDFS). Celebrated Hadoop Distributed File system is the glue that holds the big data ecosystem together. HDFS provides a robust but simple framework on top of which other software components can be built.

2.2.1 Hadoop Distributed File System

As the name suggests it is a distributed file system that can run on commodity hardware.

Design Goals of HDFS

In a large commodity hardware clusters, node failing is expected and it is the norm rather than the exception. The clusters can expand to thousands of nodes increasing the probability of some part of the cluster not working at a given time. One of the primary goals of HDFS is to be able to handle failing components and recover quickly. HDFS is targeted towards handling jobs with large data sets so efficient processing of large files is another goal. Most of the HDFS applications need write big sets of data once and read them many times.

File System

The file system abstraction provided by HDFS is a hierarchical file system like the Unix file system. There are directories and files in this system. User can put files to the HDFS, delete files, and move files and copy files like in a normal Unix file system.

HDFS Architecture

NameNode and DataNode are the main components in the HDFS architecture. NameNode is the master node that manages the file system namespace. There is one NameNode and multiple DataNodes in a HDFS deployment. NameNode can be replicated for fault tolerance. Files are split into large blocks and these blocks are stored in the data nodes. The NameNode executes file system operations like opening, closing, and renaming files and directories. It also keeps tracks of the mapping of blocks to DataNodes. For reading file a client obtains the file's block list from the NameNode and then directly talks to the DataNodes.

The NameNode keeps an edit log to track all the changes made to the file system. This edit log is saved in the NameNode's local file system. Also the NameNode keeps track of all the blocks in the file system and which data nodes these blocks are assigned to. This is called FSImage and it is also saved in the NameNode's local file system.

When the NameNode boots up, it goes into a mode called safe mode. While in this mode the DataNodes contact the NameNode to send a block report. The block report consists of the information about the blocks in the DataNode. After NameNode receives information about a configurable percentage of all the blocks in the system it goes off the safe mode and replicate the blocks that are not replicated sufficiently at that time.

While in operation, the NameNode receives heartbeats and block reports from the data nodes periodically. It uses the block report to organize the replicas of the files. The data nodes don't know anything about files and they just store blocks in its local file system. For each block a data node creates a separate file and store it. The DataNode can create directories to put the data node files but directories don't mean any hierarchy or organization.

Fault Tolerance

HDFS replicates the file blocks in different data nodes for fault tolerance. The replication policies can be configured per file in HDFS. If the replication policy is rack aware it can place the replicas in the same rack for fast access and some replicas in different racks in case the network connectivity to a rack is lost. If a data node goes down the name node can use other replicas to function properly.

2.3 Resource Management

In recent years commodity server clusters have become a cost efficient solution for running large Internet services and computationally intensive scientific computations. Frameworks like Hadoop, Storm, Giraph can run in very large clusters of commodity hardware. These frameworks were originally architected to have both cluster resource management, scheduling of tasks and running of the task in a single monolithic architecture. Because of this only one such framework can be run in a given cluster at a time because they were not aware of other frameworks in the cluster. The ability to run these frameworks alongside each other in a single cluster is beneficial both economically and in terms of operational overhead. In traditional HPC systems cluster resource managers like Torque [7], Moab [8] and SLURM [9] were used to allocate resources to the processes. The problem is different from these traditional HPC resource managers in two main aspects. First one is that the systems that run Hadoop or Storm are commodity hardware systems. The second aspect is that, just allocating the cluster resources is not sufficient for running a system like Hadoop and it requires more configuration and fine-grained application specific scheduling. To facilitate these

requirements a new resource management layer is created and the two leading projects are Apache Yarn and Apache Mesos.

2.3.1 Apache Yarn (Yet another Resource Manager)

Apache Yarn [10] provides a cluster resource management layer for better resource utilization. In the prior versions of Hadoop it was not possible to share the same cluster among different computation frameworks or between different installations of Hadoop. Before Yarn, the only way to share a cluster was to partition the cluster into different parts and run different frameworks on these partitions. This doesn't guarantee efficient usage of the cluster resources. So with the Hadoop Version 2.0 it has introduced a resource management layer (Yarn) to handle different computational frameworks in the same cluster. Yarn can run Hadoop MapReduce, Storm like computational frameworks in the same cluster by sharing resources.

Architecture

Yarn has a ResourceManager for scheduling resources of a cluster NodeManagers that runs on computational nodes of the cluster. The ResourceManager is a pure resource scheduler. An application like MapReduce that needs to run on Yarn should implement an ApplicationMaster. The ApplicationMaster is specific to an application and knows how to run the application on the cluster resources. ApplicationMaster asks for resources from the ResoureManager to run its applications. ApplicationMaster can use different policies for allocating resources and an organization can define their own policies as well.

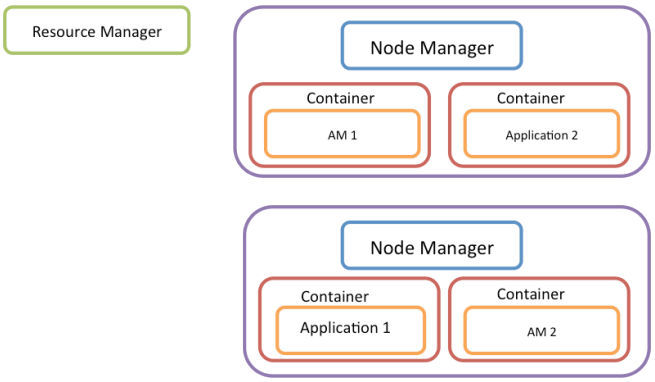

Figure 2 : Yarn Architecture

A client program submits an application to the application specific ApplicationMaster along with the resource requirement of the application. The ApplicationMaster negotiates with the resource manager to acquire the requested resources. The resources are allocated as containers. A container is essentially a set of resources allocated on a node. Now ApplicationMaster can use these containers to run the application on the nodes. To do this ApplicationMaster talks to the NodeManagers that are responsible for the allocated containers and launch the application in these nodes. After the containers are launched by the NodeManager it monitors them for resource consumption, failures etc and convey this information to the ApplicationMaster.

2.3.2 Apache Mesos

Apache Mesos [11] is a cluster resource manager which is capable of running distributed processing frameworks like Hadoop, MPI, Hypertable, Spark and Storm in a shared cluster environment. Mesos enable such sharing by managing the cluster resources among different computational frameworks.

Mesos has a master process and set of Mesos slave processes are running on the cluster nodes. The computational frameworks deployed on top of Mesos run their individual tasks on these slave nodes. Master process does the resource sharing using "resource offers" to the frameworks. A resource offer is a set of available resource for a framework. Mesos determines what resources are available for a computational framework according to the resource allocation policy and available free resources. These policies include things like fair scheduling policy, priority policies and an organization can define their own policy as well.

A framework that is running on Mesos should implement a resource scheduler and an executor process. The framework resource scheduler is registered with the Mesos master node and the master node offers resources to this resource scheduler. Executor processes are running on cluster nodes and can run individual tasks on these nodes.

Mesos has a single point of control in master node and hence it has a single point of failure. To avoid this the master node state is made soft state and if a master dies, the slave nodes can use the information in them to create another master node. Master node only keep track of active slave nodes, active frameworks and running tasks. This information can be obtained from the slave nodes in case of a master failure. In case of master failure Mesos uses ZooKeeper service to elect a new master.

2.4 NoSQL Databases

Strict structured data storages like relational database management systems don't scale well for very large data sets. NoSQL databases offer semi structured data stores for very large data sets with lower guarantees than traditional relational databases. The NoSQL databases follow two architectures in general. In the first architecture a distributed file system is used as the underlying storage mechanism for data. In this architecture the data replication and consistency is handled by the underlying file system. In the second approach a peer-to-peer approach with distributed hash tables are used. In this approach the data replication and consistency is handled by the NoSQL database. Apache HBase and Apache Accumulo are based on the DFS architecture and Apache Cassandra is based on the p2p architecture.

2.4.1 Apache HBase

Apache HBase is a column oriented, distributed, multidimensional data storage built on top of HDFS. HBase uses HDFS to store the data and on top of HDFS it provides fast record looks and update for the data tables. HBase is inspired by the Google BigTable [12] implementation.

Architecture

HBase stores the data in HDFS and relies on ZooKeeper as a distributed coordination framework. HBase architecture has a Master node and number of Region Servers. Master is responsible for managing the region servers. Each region server contains a set of regions. A region is a partition of a table. When a table is created it has only one region and belongs to one region server. When the table grows it is divided up in to multiple regions and these different regions are allocated to different region servers.

HBase Data Model

HBase data model is a multi-dimensional, distributed persisted map indexed with Row key, Column Key and a timestamp. The map is sorted with the row keys. Each row has a key and a set of column families. These column families contain the columns and columns can be added dynamically to a column family. Row keys are un-interpreted bytes. The tables are assigned to a namespace and namespace serve as a logical grouping for the tables making namespaces analogous to schemas in traditional RDBMS.

Batch Processing

HBase natively supports distributed parallel computing using Apache Hadoop because it is written on top of HDFS. This makes HBase a good choice for users who wants to run batch jobs on huge data sets. Recently Hive and HBase was integrated together making HBase data accessible through HiveQL.

Client API

HBase natively provides a client API written in Java for accessing the data store.

2.4.2 Apache Accumulo

Apache Accumulo is a column oriented, distributed, multidimensional data storage built on top of HDFS. Accumulo design is inspired by the Google Bigtable implementation and is very similar to HBase in every core aspect. There are minor detail differences in Accumulo and HBase like terminology differences, authentication mechanisms etc. But the fundamental data storage capabilities are between Accumulo and HBase are clearly similar.

Accumulo Architecture

Accumulo has a similar architecture to HBase. There is only a terminology difference between the two systems. Accumulo uses HDFS for storing the data and use ZooKeeper for distributed coordination between the nodes. Accumulo has a Master node and set of tablet servers (region servers in HBase). These tablet servers hold tablets (regions in HBase). Each table data is partitioned into tablets and stored in the tablet servers. For the logging and garbage collection Accumulo uses separate servers but HBase has these capabilities built in to the Master and Region servers.

In general Accumulo has better access control than HBase. Accumulo has cell level visibility labels and Tables ACLs. HBase doesn't have cell level visibility labels. Also Accumulo has built in support for user authentication and HBase has this support in an optional manner. Both Accumulo and HBase have the same level of scalability and performance. HBase is the more popular and widely used database among the two.

2.4.3 Apache Cassandra

Apache Cassandra is a distributed key value store developed by Facebook. Cassandra is based on a peer-to-peer distributed architecture where every node is treated as equal. Cassandra has a row oriented data model. In general Cassandra is better suited for handling large data traffic loads where real time fast random access to data is required.

Traditional RDBMS are all about preserving the ACID (A - Atomicity, C - Consistency, I - Isolation, D - Durability) properties. In reality when distributed applications are developed they need to look at 3 orthogonal properties called Consistency, Availability and Partition Tolerance. The CAP theorem says that is it not possible to preserve all three of the above properties and have an acceptable latency for the operations. So Cassandra enforces two of the above properties by loosening the bounds of another property. Cassandra values high availability and partition tolerance and sacrifices strong consistency to eventual consistency. Cassandra is influenced by the Amazon Dynamo [13] database consistency model and uses partitioned and replication similar to Dynamo. Because of the partitioning and replications applications cannot lock on the row level, which is supported in HBase.

Architecture

Cassandra is a fully distributed system with no single point of failure. Every node in the system is treated equally and performs the same operations. The nodes are arranged in a ring with P2P connections. The keys for a table are distributed among the nodes using a consistent hashing algorithm. Each node in the ring is assigned a random hash value (a value from the hash value range) and the keys belong to the particular hash value range are stored in this node. The same keys are replicated on the near nodes for high availability and this set of replicated nodes for a particular key is called the preferred list for that key. A client can contact any node in the ring to retrieve or update a value for a key. If the contacted node is not in the top preferred list for the key, the request is forwarded to a node in the preferred list. For handling consistency it uses a quorum based protocol with hinted handoff. Hinted handoff is a mechanism developed on top of quorum-based protocol to make the system working under network partitions and failures of nodes.

Data Model

Cassandra data model is borrowed from the Google Big Table design. The data model is a partitioned row store with configurable consistency. The rows are organized into tables and each row can contain any number of columns. The default API to the data is through the CQL language and there is Thrift based low level API available as well. The Thrift based API exposes the low-level storage details of Cassandra so this API is no longer recommended. The data is stored in tables and de-normalization is the norm for creating the tables. Tables are designed to suite the particular query requirements and data duplication is considered normal. In RDBMSs first a normalized data model is created and then it is de-normalized slightly to make the queries more efficient. The underlying data model in Cassandra is a row based multiple column store. These underlying details are hidden when using the CQL for manipulating the data.

Cassandra Query Language (CQL)

Cassandra provides a query language called Cassandra Query language (CQL), which is similar to SQL. The CQL doesn't support join operations because Cassandra data model doesn't support joins. The tables should be designed such that duplicating the data does not require the join operations. Cassandra support Indexes on columns for fast access the values.

Transactions

In general Cassandra doesn't support transactions for its operations. According to CAP theorem it is almost impossible to support full transactions with a very high availability and partition tolerance, which are the main goals of Cassandra. With the 2.0 release, Cassandra is supporting lightweight transactions which is a tiny subset of transactional behavior offered by RDBMs. Lightweight transactions provide the ability to compare-and-set a value in database with majority votes. It uses the PAXOS algorithm with an additional commit phase for determining whether a value exists and if it exists update another or same value.
Cassandra performance is generally greater than HBase and it scales well with the addition of more nodes.

Client API

Cassandra client API is an Apache Thrift based API. This allows the development of client libraries in various languages that have binding to Thrift. Cassandra project has a rich set of client libraries written in various programming languages making it an attractive choice in heterogeneous platforms.

Batch Processing

Cassandra has support for running MapReduce jobs on stored data using Hadoop. Also there is support for running PIG jobs and Hive queries on top of Cassandra as well. The HDFS of Hadoop is replaced with implementation called CassandraFS for running the Hadoop jobs natively on the data. Also to alleviate the heavy burdens of batch processing on Cassandra clusters, while the data is updated by the incoming traffic, concepts like virtual datacenters are introduced. In this mode the data processing happens in separate nodes that doesn't handle the incoming traffic.

2.5 Runtime Systems

Runtimes systems include the core data processing frameworks. Data processing can be done in batch mode, stream mode or as graph processing. Each of these subcategories has projects that are specialized in them. The resource managers at the lower layers manage the runtime systems in clusters. Most of these systems are already ported to run in Mesos and Yarn. The input to these systems can be coming from raw data in HDFS as well as structured data in NoSQL databases. Output of these runtime systems can also be directed to these storage layers. The stream frameworks usually read data streams from queues like Kafka [14], Krestel [15] or traditional ActiveMQ like queues.

Some of these systems are tightly integrated with the underlying HDFS layer. For example Hadoop MapReduce, Hama and Giraph are tightly integrated with the HDFS.

2.5.1 Batch Processing

The bulk of the large-scale data processing frameworks are batch-processing frameworks. Today the dominant processing model for large data sets is MapReduce. Even though MapReduce has become the dominant processing model for large-scale data processing, it is not suitable for all types of batch processing tasks. To overcome some of these disadvantages new processing models like in Apache Spark [16] has and Apache Hama [17] has being introduced. Some of these processing models are new and it still early to see weather they are successful or not, but from a theoretical standpoint they have very nice concepts and approaches to the problem.

2.5.2 Hadoop MapReduce (Prior to version 2.0)

Hadoop MapReduce framework has three main parts. They are Hadoop Distributed file System, JobTracker and TaskTracker. The MapReduce framework uses HDFS for sharing data among the computation tasks of a map reduce job.

A client submits a job to the JobTracker. JobTracker runs the job by submitting the Map tasks and Reduce tasks of the job to the TaskTracker. TaskTrackers run on the compute nodes of the cluster. A TaskTracker has slots available for running the jobs. JobTracker pushes the individual tasks to TaskTrackers with free slots.

If a TaskTracker fails to finish a task the JobTracker can re-schedule the task using another TaskTracker. First the JobTracker reads the input to the Job from the HDFS. Then it splits this input partitions and run Map tasks on each of the partitions. The JobTracker tries to run the Map jobs to the data as close as possible. The Map tasks stores the intermediate results in the local files system that it runs. After all the Map tasks are finished the output of the Map operations are sorted in disks. Then these outputs are assigned to reduce tasks and they read the data and run the reduce code. The data emitted from the reduce phase is saved in the HDFS.

2.5.3 Hadoop MapReduce Version 2.0

The Hadoop MapReduce version 2 doesn't include the JobTracker and TaskTracker from the previous version. The new MapReduce runs on top of Apache Yarn and uses its functionalities for the resource allocation and scheduling. The MapReduce part of the JobTracker is replaced by the MapReduce Application master for Apache Yarn. The Yarn node managers replace the TaskTrackers. JobHistory server is a new addition to the architecture that keeps track of the finished jobs. The new architecture is shown in figure 3.

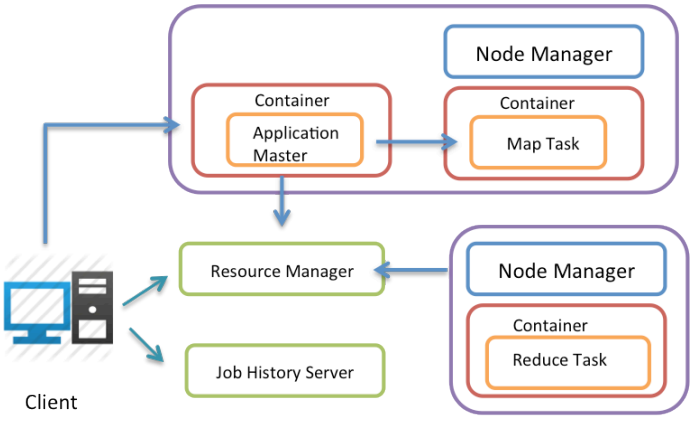

Figure 3 MapReduce V.2

Unlike the previous JobTracker the new MapReduce Application Master is created per job basis. When the job finishes the resources are de-allocated and MapReduce Application master finishes. The clients first submit its job requirements to the Yarn Resource Manager to allocate the resources for the job. The resource manager allocates a container to run the MapReduce Application Master for the job and start the application master. After the MapReduce Application Master is started, it can ask for the required resource from the ResourceManager to run the MapReduce

job. After the required resources are allocated MapReduce Application master runs the map tasks and reduce tasks on these resources.

2.5.4 Apache Spark

Apache Spark is a distributed in-memory parallel computing framework developed by UC Berkeley and now it is available as an open source project in ASF. The goal of the project is to build a faster distributed parallel processing framework that can perform better for certain tasks than Hadoop. The project is targeting iterative algorithms and interactive analytics problems on top of large data sets. Even though the project is targeting these applications it can run general map reduce jobs efficiently as well.

Apache Spark is built around a concept called Resilient Distributed Datasets (RDD) [18]. RDDs are distributed in memory fault tolerant data structures that allow users to persist them and partition them for distributed parallel computations. RDDs are associated with rich set of operators so that user can manipulate them for their computations. An operator is applied on RDD and the result is again a RDD. The lineage information about the RDD's are preserved so that in case of a node failure a lost RDD can be reconstructed by using the lineage information.

A distributed computation in Spak can be viewed as a graph that change the state of the RDDs. Initially RDD is created from the user data. Spark can use local files systems, network protocols or HDFS to get the initial data into the system. Once an RDD is created from the input data, user can apply various operators on this RDD to perform the calculations. The RDDs produced as a result can be used for later calculations. At any point RDD's can be persisted to memory. Once a failure happens the lineage graph along with the input data can be used to reproduce the lost RDDs. For calculations that involve deep and wide lineage graphs it is not efficient to calculate the lost RDDs from the input data. If the user has saved the intermediate RDDs to persistent storage this information can be used without starting from the beginning of the graph. Spark is about 10x to 100x faster than Hadoop for certain type of computations like in iterative computations.

Steaming

Recently Spark has introduced a streaming layer on top of the core functionalities. The stream processing engines like Storm and S4 doesn't provide exactly one guarantee for message processing. Instead they provide at least once guarantee for message processing. Spark uses the same RDD model they use for batch processing for doing stream processing. In this case instead of reading the input data from the file system or HDFS Spark reads the input from an stream source like a message queue. Then it creates RDDs fro the incoming message data and use the RDDs for computations.

2.5.6 Distributed Stream Processing

Apache has two major open source distributed stream processing engines. They are Apache Storm and Apache S4 [19]. Apache Storm is a real time stream-processing framework built by Twitter and now available as an Apache project.

S4 stands for Simple Scalable Streaming System and it was developed by Yahoo and donated to Apache Software Foundation in 2011. S4 is a fully distributed real time stream-processing framework. It employs the Actors model for computations. The processing model is inspired by MapReduce and uses key value based programming model as in MapReduce.

2.5.7 Graph Processing

Sequence of MapReduce jobs is the traditional way of doing graph processing. Doing graph processing on top of MapReduce is extremely in-efficient due to the fact that at each map reduce phase the entire graph has to be read into memory, sort and shuffle this graphs and then write the entire updated graph into disk.

2.5.8 Apache Giraph

Apache Giraph is a graph-processing framework built on top of Apache Hadoop. Giraph is based on the large-scale distributed Graph processing system called Pregel [20] by Google. Giraph is based on the bulk synchronous computation model by Valiant [21]. Pregel is built on top of Apache Hadoop and a Pregel computation is run as a Hadoop MapReduce job. This MapReduce job consists only of Map tasks. The Pregel tasks are run inside these Map tasks. Pregel uses Apache ZooKeeper for coordination among its tasks and Netty [22] as the messaging system for the inter node communications.

Set of Vertices and Edges represent a graph in Giraph and these vertices and edges are read through the Hadoop. Vertices are the first class citizens that do the computation and Edges are there to represent the connections. The computation is carried according to bulk synchronous parallel model in super steps. In each super step all the vertices does computation and then communicate with each other through messaging. After the messaging there is synchronization among all the nodes and then again the next super step is executed. Apache Hadoop is used as a data distribution and parallel task execution for Pregel and Apache Yarn replaces this layer in the newer versions of Pregel. Pregel can be used for extremely large-scale graphs and recently there has being claims of it running for graphs with trillion edges [23].

2.6 High Level Languages for Data Processing

The APIs provided by the processing engines are too low level for most of the users who want to analyze the data stored in large-scale storage systems quickly. In most of the organizations where Hadoop is used for processing large amounts of data, Hadoop is used as a data storage system as well. Because of the large amount of data accumulated in HDFS, the programmers are accustomed to write small batch jobs to retrieve the information stored in HDFS. This is true for NoSQL databases as well. The high level languages provide easy access to such data stored in HDFS and NoSQL databases. Once written and executed they usually compiled in to MapReduce jobs and run on the data stored.

2.6.1 Apache PIG

PIG is a high level programming language for composing Hadoop MapReduce jobs. The programming language is created by Yahoo to facilitate the quick development of Hadoop Jobs and the language itself is named Pig Latin [24]. The most popular language for querying data is SQL and it is a declarative language. PIG is a procedural language. Because PIG is procedural it can express the data pipelines in a more natural way. Usually a data pipeline for an application includes taking data from one or more sources, cleansing it and do some initial transformations to make it readable by the applications and store it in data store. These types of operations are procedural in nature and can be modeled easily with a procedural language.

It is important to store data in between pipeline operations for check pointing purposes. Check pointing allows the data pipelines to be re-run from the middle in case of failures. Pig Latin allows the programmers to checkpoint at any point in the execution. In declarative languages user is forced to checkpoint only when the semantics of such languages allows for check pointing. One of the advantages of PIG over a declarative language is that the programmer is in control of how the operations are acting over the data. In declarative languages the programmer is relying on the query optimizer to make the right choice for the queries. The downside to this approach is that the programmer has to be more knowledgeable about data operations and algorithms. Because of the procedural nature of PIG Latin it is best suitable for data pipeline applications, iterative data processing and data transformation tasks. All the above tasks are dealing with unstructured data. So PIG is best suitable for processing unstructured data.

2.6.2 Apache Hive

Apache Hive [25] is a high level language for large-scale data analytics on top of Apache Hadoop. Hive provides a mechanism to project structure onto the data stored in HDFS and query the data using a SQL-like language called HiveQL. HiveQL is a declarative language similar to SQL. In a declarative language like SQL, user defines what data

they want and not how to retrieve the data. Because of this Hive is best suitable for running queries on data warehouses where data is pre prepared and in a nice readable format. In other terms Hive is best suitable for processing structured data.

Hive can create a table structure on top of the data in HDFS and use this structure for SQL queries. The SQL queries are compiled in to map reduce jobs and run on the data-using Hive. For some simple queries Hive can directly read data from HDFS without running MapReduce jobs but this is not the normal case. Hive uses an RDBMS for keeping the metadata about the tables it create on top of HDFS data. By default Hive runs with embedded Derby and it supports MySQL as well. The tables created by Hive can be accessed through JDBC using the Hive JDBC driver making the HIVE tables accessible through the rich set of JDBC operations supported by Java.

2.6.3Apache MRQL

Apache MRQL [26] pronounced, as Apache Miracle is a latest addition to the family of languages developed for large-scale distributed data analysis. The MRQL language is similar to SQL. MRQL supports data analytics on top of Hadoop, Hama and Spark.

2.7 Data Analytics

Standard data analytics frameworks on top of big data solutions were lacking in the big data stack. Data analytics on top of the big data solutions were built for specific applications and standard reference implementations were hard to find. Apache has several projects aiming at implementing standard data analytics frameworks on top of big data using machine learning algorithms.

2.7.1 Apache Mahout

Apache Mahout is a project aiming at building scalable machine learning algorithms for large data sets. The library primary focuses on MapReduce programs developed on top of Apache Hadoop but the Mahout project is not restricted to Hadoop based algorithms and looking to incorporate machine-learning algorithms on top of other distributed platforms as well. The project is under development and lots of machine learning algorithms are being development at the moment. The project includes algorithms for following broader tasks.

Task	Algorithms
Classification	Boosting, Neural Networks, Logistic Regression, Naive Bayes
Clustering	Canopy Clustering, K-Means, Fuzzy K-Means, Mean Shift Clustering, Hierarchical Clustering, Dirichlet Process Clustering, Latent Dirichlet Allocation, Spectral Clustering, Minhash Clustering, Top Down Clustering
Pattern Mining	Frequent Item Mining
Regression	Work in progress
Dimension Reduction	Work in progress

2.7.2 MLlib

MLlib is a machine learning algorithm library implemented on Apache Spark. The library supports binary classification, regression, clustering and collaborative filtering, as well as an underlying gradient descent optimization primitive. Here is a summary of the algorithms supported in each category.

Task	Algorithms
Binary classifications	Linear support vector machines, Logistic Regression
Regression	Linear regression, L1 (lasso) regression, L2 (ridge) regularized.
Clustering	K-means,
Collaborative filtering	Alternating Least Squares

3. Conclusion

This report analyzed the Apache Open Source big data projects by putting them in to a layered architecture and investing the interrelationships among the different layers and the projects within them. The projects that are discussed in this report are mostly autonomous projects sometimes backed by competing technology companies. Still these projects have found ways to co-exist complimenting each other to create a unique open environment for innovative product development in the big data space. The number of projects entering the Apache is steadily increasing with many important open projects finding Apache as a permanent home for their success.

We identified six layers for the big data stack in Apache Software foundation and the layers at the top depend on the layers at the bottom. Few years ago we couldn't identify such clear layers in the big data projects. Some projects listed in this report are still not using the full power of this layered architecture. But the projects are evolving very fast and the general directions they evolve suggest that these projects will leverage the benefits of this layered architecture over time.

4. References

[1] Apache Software Foundation. [Online]. http://www.apache.org/

[2] Netflix. Netflix Open Source Software Center. [Online]. http://netflix.github.io/#repo

[3] Linkedin. Open Source @LinkedIn. [Online]. http://data.linkedin.com/opensource

[4] Jeffrey, and Sanjay Ghemawat Dean, "MapReduce: simplified data processing on large clusters," in *Communications of the ACM 51.1 (2008): 107-113*.

[5] Sanjay, Howard Gobioff, and Shun-Tak Leung Ghemawat, "The Google file system.," in *ACM SIGOPS Operating Systems Review. Vol. 37. No. 5*, 2003.

[6] Patrick, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Hunt, "ZooKeeper: wait-free coordination for internet-scale systems.," in *In Proceedings of the 2010 USENIX conference on USENIX annual technical conference, vol. 8, pp. 11-11*, 2010.

[7] Torque Resource Manager. [Online]. http://www.adaptivecomputing.com/products/open-source/torque/

[8] Moab. [Online]. http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-basic-edition/

[9] Slurm Workload Manager. [Online]. http://slurm.schedmd.com/

[10] Apache Software Foundation. (2013, Oct.) Apache Software Foundation. [Online]. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[11] Benjamin, et al. Hindman, "Mesos: A platform for fine-grained resource sharing in the data center.," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation.*, 2011.

[12] Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Chang, "Bigtable: A distributed storage system for structured data.," in *ACM Transactions on Computer Systems (TOCS) 26, no. 2 (2008): 4.*

[13] Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. DeCandia, "Dynamo: amazon's highly available key-value store.," in *In SOSP, vol. 7, pp. 205-220*, 2007.

[14] Jay, Neha Narkhede, and Jun Rao Kreps, "Kafka: A distributed messaging system for log processing.," in *In Proceedings of the NetDB*, 2011.

[15] Kestrel. [Online]. https://github.com/robey/kestrel

[16] Matei, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica Zaharia, "Spark: cluster computing with working sets," in *In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, pp. 10-10*, 2010.

[17] Sangwon, Edward J. Yoon, Jaehong Kim, Seongwook Jin, Jin-Soo Kim, and Seungryoul Maeng Seo, "Hama: An efficient matrix computation with the mapreduce framework," in *In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, pp. 721-726.* , 2010.

[18] Matei, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica Zaharia, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pp. 2-2*, 2012.

[19] Leonardo, Bruce Robbins, Anish Nair, and Anand Kesari Neumeyer, "S4: Distributed stream computing platform.," in *In Data Mining Workshops (ICDMW), 2010 IEEE International Conference on, pp. 170-177*, 2010.

[20] Grzegorz, Matthew H. Austern, Aart JC Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Malewicz, "Pregel: a system for large-scale graph processing.," in *In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 135-146*, 2010.

[21] Leslie G. Valiant, "A bridging model for parallel computation," in *Communications of the ACM 33, no. 8 (1990): 103-111*.

[22] Netty Project. [Online]. http://netty.io/

[23] Scaling Apache Giraph to a trillion edges. [Online]. https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920

[24] Christopher, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Olston, "Pig latin: a not-so-foreign language for data processing.," in *In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1099-1110*, 2008.

[25] Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Thusoo, "Hive-a petabyte scale data warehouse using hadoop," in *In Data Engineering (ICDE), 2010 IEEE 26th International Conference on, pp. 996-1005*, 2010.

[26] Leonidas, Chengkai Li, and Upa Gupta. Fegaras, "An optimization framework for map-reduce queries," in *In Proceedings of the 15th International Conference on Extending Database Technology, pp. 26-37*, 2012.

[27] Reynold, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Xin, "Shark: SQL and rich analytics at scale," in *arXiv preprint arXiv:1211.6176 (2012)*.