ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 11, May 2017

YARN in interactive Cross - Platform Query System

S.Jancy Sickory Daisy, D.Mirunalini
Assistant Professor in Computer Science and Engineering Department
P.R.Engineering College, Vallam, Thanjavur, TamilNadu, India.

Abstract* - Now a day's storage of data has a rapid growth which leads to new technologies in the field of big data. IDC predicts that the data volume worldwide will reach 40ZB by 2020. Many systems have been developed for managing and analyzing PB level structured data. Major big data stacks often consist of a general purpose, multi-staged computation framework (e.g. Hadoop) and an SQL query system (e.g. Hive) on its top system. Existing mining and analysis techniques simply are not up to the task of handling big data. One possible solution to this problem is to build Hadoop clusters. In this paper, a new technique has been proposed to manage large scale storage using Hadoop clusters – YARN (Yet Another Resource Negotiator). This system architecture mainly includes three layers: (1) a storage layer; (2) a scheduling and execution layer; and (3) an application layer providing a cross-platform query interface. Hadoop cluster's parallel processing capabilities certainly help with the speed of the analysis, but as the volume of data to be analyzed grows the cluster's processing power may become inadequate.

***Index Terms* - Big data, cross platform, Hadoop cluster, Hive.**

I. INTRODUCTION

Analyzing massive amounts of data and obtaining valuable information and knowledge, researchers have developed many excellent systems and technologies A common model is placing a SQL query system over a general purpose distributed computing framework, such as Hive over Hadoop , Tenzing over Google MapReduce and Shark over Spark. As we have tremendous growth in social network like Google, Facebook and E-commerce high relevance and coupling degree of data is necessary, that results in the rapid growth of the data to PB level and above. Supporting interactive query on such large volumes of data necessitates the development of large-scale storage management ability and rapid analysis and calculation capability.

These requirements lead to pose new challenges for both relational database and big data processing technology. For structured data, relational database is undoubtedly the most classic and popular database system, such as Oracle, MySQL, SQLServer, and DB2. In 1970, Codd first proposed a new model of the relationship, which started the research on the relational method and theory of database. After decades of development, relational database has come to be widely used in various types of information management systems and business application systems, and has become an effective storage and analysis tool for data warehouse.

At present, parallel database based on Massively Parallel Processing (MPP) architecture can manage hundreds of TB of data. Such database system consists of many loosely coupled processing units, and each unit has its own private computing and storage resources, such as CPU, cache, memory, hard disk, and operating system. The most significant features of MPP database are shared- nothing and multiple copies of data. Therefore, SQL commands can be split and scheduled to different processing units to be executed concurrently so as to achieve better performance. 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.

A 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 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 JobTracker and one lave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master. Minimally, applications specify the input/output locations and supply ma*p* and reduc*e* 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 JobTracker 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.The existing mining and analysis techniques are not up to the task of handling big data. With technology boom and digitalization, huge volume of data started jamming up the servers.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 11, May 2017

One possible solution to this problem is to build Hadoop clusters. Contextually, Hadoop - an open-source framework, initiated speedy processing of BIG data, thereby allowing effective Data Mining, Reporting, and Predictive Analysis. A Hadoop cluster is a special type of cluster that is specifically designed for storing and analyzing huge amounts of unstructured data. A Hadoop cluster is essentially a computational cluster that distributes the data analysis workload across multiple cluster nodes that work to process the data in parallel. By using HDFS and clusters, this system effectively integrates large-scale storage management with interactive query and analysis. We propose to implement a cross-platform query interface, through clusters in which clients can execute online queries on relational databases.

II. RELATED WORK

The Map-Reduce paradigm native to Apache Hadoop is an effective tool to pre-process unstructured and semi-structured data sources such as images, text, raw logs, XML/JSON objects etc. Hive in Hadoop, map files onto a database table and provide an SQL query interface. It converts SQL statements into a series of MapReduce tasks.When a SQL query is issued to Hive, it is translated into one or multiple MapReduce jobs. During translation, four kinds of informations are obtained in Hive: schema of shuffled data, compression algorithm applied to each column, predicates and join conditions.

This information is then stored in the MapReduce job configuration file, which is shared by map/reduce tasks across by clusters. MPP DBMSs are the database management systems built on top. In these systems each query is split into a set of coordinated processes executed by the nodes of MPP grid in parallel, splitting the computations the way they are running times faster than in traditional SMP RDBMS systems.

It can easily scale the grid by adding new nodes into it. To be able to handle huge amounts of data, the data in these solutions is usually split between nodes (shared) the way that each node processes only its local data. Dremel is a way of analyzing information. Running across thousands of servers, it “query” large amounts of data, such as a collection of web documents or a library of digital books or even the data describing millions of spam messages. This is akin to analyzing a traditional database using SQL, the Structured Query Language that has been widely used across the software world for decades.

Shark is open source and compatible with Apache Hive, and has already been used at web companies to speed up queries by 40–99%. Shark builds on a recently-proposed distributed shared memory abstraction called Resilient Distributed Datasets (RDDs) to perform most computations in memory while offering fine-grained fault tolerance.

In-memory computing is increasingly important in large-scale analytics for two reasons. (1) Many complex analytics functions, such as machine learning and graph algorithms, are iterative, going over the data multiple times; thus, the fastest systems deployed for these applications are in-memory. (2) Even traditional SQL warehouse workloads exhibit strong temporal and spatial locality, because more-recent fact table data and small dimension tables are read disproportionately often.

MapR hadoop distribution works on the concept that a market driven entity is meant to support market needs faster. Leading companies like Cisco, Ancestry.com, Boeing, Google Cloud Platform and Amazon EMR use MapR Hadoop Distribution for their Hadoop services. Unlike Cloudera and Horton works, MapR Hadoop Distribution has a more distributed approach for storing metadata on the processing nodes because it depends on a different file system known as MapR File System (MapRFS) and does not have a Name Node architecture. MapR hadoop distribution does not rely on the Linux File system.

Hortonworks, founded by Yahoo engineers, provides a ‘service only’ distribution model for Hadoop. Hortonworks is different from the other hadoop distributions, as it is an open enterprise data platform available free for use. The distribution provides open source platform based on Apache Hadoop for analysing, storing and managing big data. The engineers of Hortonworks are behind most of Hadoop’s recent innovations including Yarn, which is better than MapReduce in the sense that it will enable inclusion of more data processing frameworks.

Spark was introduced by Apache Software Foundation for speeding up the Hadoop computational computing software process. Spark is not a modified version of Hadoop and is not, really, dependent on Hadoop because it has its own cluster management. Hadoop is just one of the ways to implement Spark.

It uses Hadoop in two ways – one is storage and second is processing. Since Spark has its own cluster management computation, it uses Hadoop for storage purpose only. Impala is integrated with Hadoop to use the same file and data formats, metadataImpala brings scalable parallel database technology to Hadoop, enabling users to issue low-latency SQL queries to data stored in HDFS and Apache HBase without requiring data movement or transformation, security and resource management frameworks used by MapReduce, Apache Hive, Apache Pig and other Hadoop software.

Impala is promoted for analysts and data scientists to perform analytics on data stored in Hadoop via SQL or business intelligence tools. The result is that large-scale data processing (via MapReduce) and interactive queries can be done on the same system using the same data and metadata – removing the need to migrate data sets into specialized systems and/or proprietary formats simply to perform analysis.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 11, May 2017

III. SYSTEM ARCHITECTURE

In figure 1 the core component of the system consists of into three main layers according to logic functions: the storage layer, scheduling and execution layer, and application layer. These layers are packed into middleware, which can work on other systems with minimal changes. The storage layer is constructed using HDFS. It stores and manages PB level data with features such as high scalability, compatibility, and fault-tolerance. The storage layer contains three important interface (1) interface used for providing the data block distribution information of the file to the scheduler module through NameNode; (2) read/write interface of local data to the query engine module; (3) the read/write interface of HDFS to the ETL module. Notably, we do not make any changes to the interfaces of the storage layer, but only use the standard API functions provided by HDFS. The other modules read/write data from/to HDFS actively by calling these API functions.

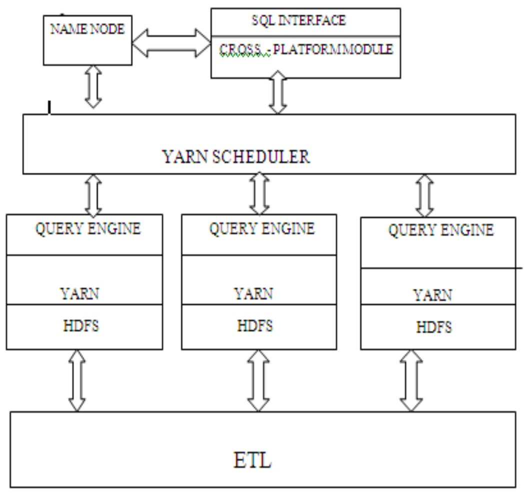

Fig 1 System Architecture

The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (*RM*) and per-application Application Master (*AM*). An application is either a single job or a DAG of jobs.

The Resource Manager and the NodeManager form the data-computation framework. The Resource Manager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage CPU, memory, disk, and network and reporting the same to the Resource Manager/Scheduler.

Resource Manager has two main components:

- *Scheduler*
- *Applications Manager*:

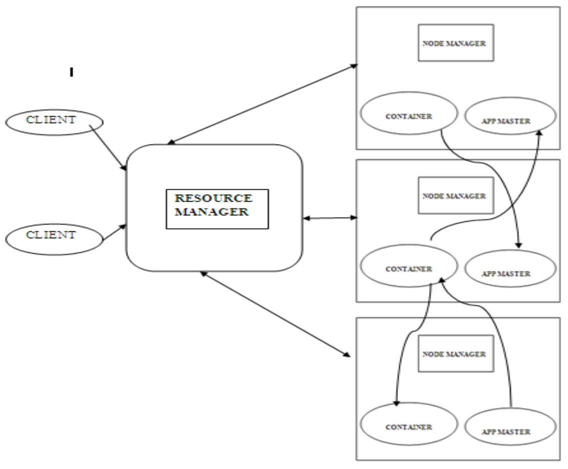

Fig 2 Resource Manager

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based the resource requirements of the applications; it does so based on the abstract notion of a resource Containe*r* which incorporates elements such as memory, CPU, disk, network etc. The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the Capacity Scheduler and the FairScheduler would be some examples of plug-ins.

The Applications Manager is responsible for accepting job-submissions, negotiating the first container for executing the application specific Application Master and provides the service for restarting the Application Master container on failure. The per-application Application Master has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

IV. YARN Framework

YARN framework figure 3, is responsible for doing Cluster Resource Management. Multiple applications can run on Hadoop via YARN and all application could share common resource management. Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs. Multiple applications can run on Hadoop via YARN and all application could share common resource management. Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs.

The original incarnation of Hadoop closely paired the Hadoop Distributed File System (HDFS) with the batch oriented MapReduce programming framework, which handles resource management and job scheduling on Hadoop systems and supports the parsing and condensing of data sets in parallel.

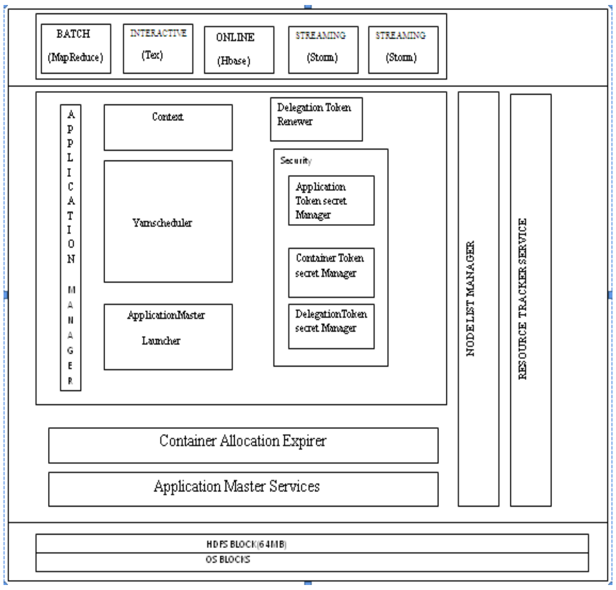

Fig 3: YARN Framework

The core of the Resource Manager- Applications Manager: Responsible for maintaining a collection of submitted applications. Also keeps a cache of completed applications so as to serve users' requests via web UI or command line long after the applications in question finished.

Application Master Launcher: Maintains a thread-pool to launch AMs of newly submitted applications as well as applications whose previous AM attempts exited due to some reason. Also responsible for cleaning up the AM when an application has finished normally or forcefully terminated.

Yarn Scheduler: The Scheduler is responsible for allocating resources to the various running applications subject to constraints of capacities, queues etc. It performs its scheduling function based on the resource requirements of the applications such as memory, CPU, disk, network etc. Currently, only memory is supported and support for CPU is close to completion.

Delegation Token Renewer: In secure mode, RM is Kerberos authenticated and so provides the service of renewing file-system tokens on behalf of the applications. This component renews tokens of submitted applications as long as the application runs and till the tokens can no longer be renewed.

Token Secret Managers (for security): Resource Manager has a collection of Secret Managers which are charged with managing tokens, secret-keys that are used to authenticate/authorize requests on various RPC interfaces.

Application Token Secret Manager: To avoid arbitrary processes from sending RM scheduling requests, RM uses the per-application tokens called Application Tokens. This component saves each token locally in memory till application finishes and uses it to authenticate any request coming from a valid AM process.

Container Token Secret Manager: Secret Manager for Container Tokens that are special tokens issued by RM to an AM for a container on a specific node. Container Tokens are used by AMs to create a connection to the corresponding NM where the container is allocated. This component is RM-specific, keeps track of the underlying master and secret-keys.

RM Delegation Token Secret Manager: A Resource Manager specific delegation-token secret-manager. It is responsible for generating delegation tokens to clients which can be passed on to unauthenticated processes that wish to be able to talk to RM.

Container Allocation Expirer: This component is in charge of ensuring that all allocated containers are used by AMs and subsequently launched on the correspond NMs. AMs run as untrusted user code and can potentially hold on to allocations without using them, and as such can cause cluster under-utilization. To address this, Container Allocation Expirer maintains the list of allocated containers that are still not used on the corresponding NMs. For any container, if the corresponding NM doesn't report to the RM that the container has started running within a configured interval of time, by default 10 minutes, the container is deemed as dead and is expired by the RM

Resource Tracker Service: This is the component that responds to RPCs from all the nodes. It is responsible for registration of new nodes, rejecting requests from any invalid/decommissioned nodes, obtain node-heartbeats and forward them over to the Yarn Scheduler. It works closely with NM Liveliness Monitor and Nodes List Manager.

Nodes List Manager: A collection of valid and excluded nodes. Responsible for reading the host configuration files specified via **yarn. Resource manager. Nodes. Include-path** and **yarn. resource manager . Nodes. Exclude - path** and seeding the initial list of nodes based on those files. Also keeps track of nodes that are decommissioned as time progresses.

V. CROSS -PLATFORM QUERY

Big data applications often need to access datasets on different platforms that may even be cross-domain. For structured data, the time cost of data extraction and loading cannot meet the real-time requirement. The different platforms involved interconnect via LAN or Internet, resulting in a distributed and heterogeneous network topology. In this network topology, the data sources are dynamic, heterogeneous, and autonomous.

Figure 4, the cross-platform query interface contains three main components: SQL interface, cross-platform module, and global table. The SQL interface provides a command shell for users and forwards query commands to the cross- platform module. If a request command involves several datasets on different platforms, the cross-

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 6, Issue 11, May 2017

platform module queries the global table and gets the information of Location (a data structure). Then, it splits the command according to the variable tag name of Location, sends the sub-command to the slave platform as master, and receives the result.

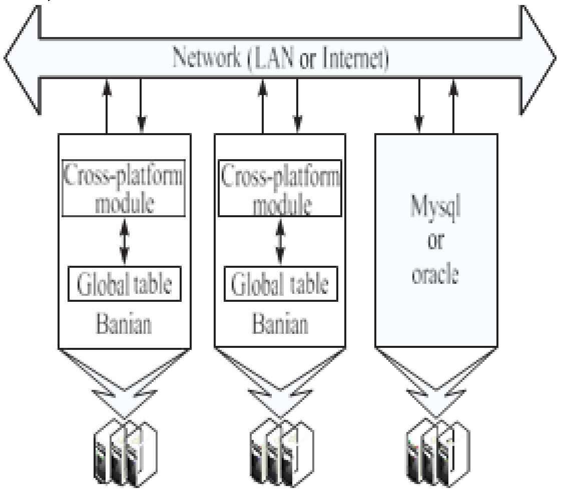

Fig 4 Cross-Platform topology

The global table stores the configuration information of all platforms using a data structure called Geoinfosys.

```
struct Geoinfosys
        char *giname;
        char *home;
        int     portid;
        int  authority;
        char *user;
        char *pwd;
```

Let us look at the cross-platform query workflow by taking the execution of a join query. We deploy two systems called GIS1 and GIS2, with the following values of Geoinfosys structure:

```
GIS1 _Geoinfosys
giname=‘ GIS1’;
home=‘166.111.134.49’;
  portid=4326; authority=1;
GIS2 _Geoinfosys
giname=‘ GIS2’;
home=‘166.111.135.49’;
portid=4326;;
authority=1;
```

We create database db1 on GIS1 and add table weblog, and create database db2 on GIS and add table userinfo. The cross-platform module splits the SQL command into sub-commands C1 and C2. Send command C1 to banian2 Location.host, wait for banian2 to return the result of C1.

Store the result of C1 and execute command C2.

Note that clients should add the platform name as a prefix and underline the connection with the database

Select * from GIS1 db1.Weblog command join GIS2 db2.userinfo on GIS1db1.weblog.sourceip = GIS2 db2.userinfo.sourceip where GIS1 db1.weblog.time >1401552000 And GIS2 db2.userinfo.zipcode = 100084

```
C1          Select        GIS2  db2.userinfo.sourceip
from
            GIS2 db2.userinfo Where
            GIS2 db2.userinfo.zipcode = 100084.
C2          Select  *  from GIS1 db1.weblog where
            GIS1db1.weblog.sourceip =  Fresultg and
GIS1
            db1.weblog.time>1401552000.
```

VI. CONCLUSION

The primary benefit to using Hadoop clusters is that they are ideally suited to analyzing big data. Big data tends to be widely distributed and largely unstructured. The data does not have to be uniform because each piece of data is being handled by a separate process on a separate cluster node. The proposed system is to implement Hadoop clusters to store huge data and analyze data in clusters. The disadvantage to using a Hadoop cluster is that the clustering solution is based on the idea that data can be "taken apart" and analyzed by parallel processes running on separate cluster nodes. If the analysis cannot be adapted for use in a parallel processing environment, then a Hadoop cluster simply is not the right tool for the job. Probably the most significant drawback to using a Hadoop cluster is that there is a significant learning curve associated with building, operating and supporting the cluster. Unless you happen to have a Hadoop expert in your IT department, it is going to take some time to learn how to build the cluster and perform the required data analysis.

VII. FUTURE ENHANCEMENT

In YARN, the Resource Manager is primarily limited to scheduling i.e. only arbitrating available resources in the system among the competing applications and not concerning itself with per-application state management. Allocating cluster resources for different applications in a multi-tenant cluster can be achieved using resource schedulers – CapacityScheduler(sharingcluster resource) and Fair Scheduler

REFERENCES

[1]. Tao Xu, Dongsheng Wang×, and Guodong Liu Banian: A Cross-Platform Interactive Query System for Structured Big Data, Vol 20, No 1, pp 62- 71,2015

[2]. S. Ghemawat, H. Gobioff, and S. T. Leung, The Google file system, ACM SIGOPS Operating Systems Review, vol. 37, no. 5, pp. 29–43, 2003.

[3]. J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, Commun. of ACM, vol. 51, no. 1, pp. 107–113, 2008.

[4]. K. Shvachko, H. Kuang, S. Radia, and R. Chansler, The Hadoop distributed file system, in Proceedings of IEEE Conference on Mass Storage Systems and Technologies (MSST), 2010, pp. 1–10.

[5]. HBase project, http://hbase.apache.org/, 2014.

[6]. Greenplum Inc., Greenplum Database: Powering the data-driven enterprise, http://www.greenplum.com/resources,.

[7]. Impala project, http://impala.io/, 2014.

[8]. S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, BlinkDB: Queries with bounded errors and bounded response times on very large data, in Proceedings of the 8th ACM European Conference on Computer Systems, New York, NY, USA, 2013, pp. 29–42.