Imperial Journal of Interdisciplinary Research (IJIR)
Vol-2, Issue-7, 2016
ISSN: 2454-1362, http://www.onlinejournal.in

Big data on Cloud using Hadoop

Sandeep Gade, Aateef Pathan, Shashank Tomar & Sahshanu Razdan
Sinhgad Academy of Engineering

Abstract*: Organizations that are working on the traditional systems like Oltp & Olap model are facing many challenges like the limitations of the resources which includes govern ace of the system-level and hardware software network storage However we are focusing on the Cost Time and the Speed of the big data project in the cloud and we will show a way in which dead data in the archrivals can be pipelined for the processing using mapreduce algorithm, our API will be unique till dates.*

Introduction

Big Data as high volume, velocity and variety information assets that demand low on cost, innovative forms of information processing for enhanced insight and decision making.
Big data has also been defined by the three Vs:

Volume: Big data requires processing high volumes of low-density, unstructured Apache Hadoop data—that is, data of unknown value, such as click streams on a web page and a mobile app, network traffic, sensor-enabled equipment capturing data at the speed of light, and many more.

Velocity: The fast rate at which data is received and perhaps acted upon. The highest velocity data normally streams directly into memory versus being written to disk. Some Internet of Things (IoT) applications have health and safety ramifications that require quick evaluation and action.

Variety: Structured, Unstructured and Semi-structured data types, such as text, audio, CRM, ERP, JSON files require additional processing to both derive meaning and the supporting metadata. Once understood, unstructured data has many of the same requirements as structured data, such as summarization, lineage, auditability, and privacy. Further complexity arises when data from a known source changes without notice. Frequent or quick schema changes are an enormous burden for both transaction and analytical environments.

The Apache Hadoop system is a challenging environment to work with, but cloud deployments introduce additional levels of complexity because of the constraints (and freedoms) offered by the cloud environment.

Clouds provide a homogeneous operating environment (for instance, identical operating system (OS) and libraries on all cloud nodes, possibly via virtualization).
The advantages and disadvantages of cloud deployments can work for and against the use of Apache Hadoop within these environments, depending upon the exact cloud service. The restrictions and limitations of a private cloud service differ enormously, compared to a public cloud service.
Big-Data on cloud using Apache Hadoop consists following components:

Warehousing as a Service: Warehousing as a Service is a fast, fully managed, petabyte-scale data warehouse that makes it simple and low on cost to analyze all your data using your existing business intelligence tools. Warehousing as a Service delivers fast query performance by using columnar storage technology to improve throughput and parallelizing queries across multiple nodes.

Cloud Mapreduce: It simplifies Big-Data processing, providing a managed Apache Hadoop framework that makes it easy, fast, and low on cost for you to distribute and process vast amounts of your data across dynamically scalable compute instances.

Object Storage: It provides developers and IT teams with secure, durable, highly-scalable object storage. It is easy to use, with a simple web service interface to store and retrieve any amount of data from anywhere on the web.

Relational Database System: It makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you up to focus on your applications and business.

NoSQL Database: It is a fast and scalable service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models. Its scalable data model and reliable performance make it a great fit for mobile,

web, gaming, ad tech, IoT, and many other applications.

Visualization Tool: It is a cloud-powered business intelligence (BI) service that makes it easy for all employees to build visualizations, perform ad-hoc analysis, and quickly get business insights from their data.

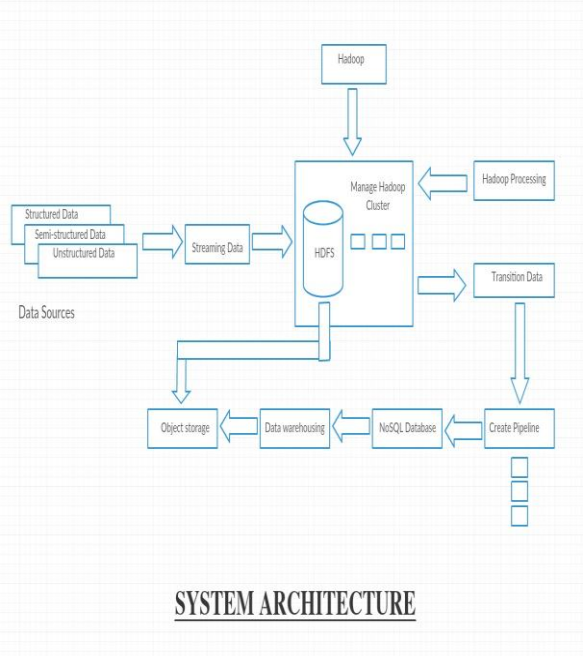

Figure: 1

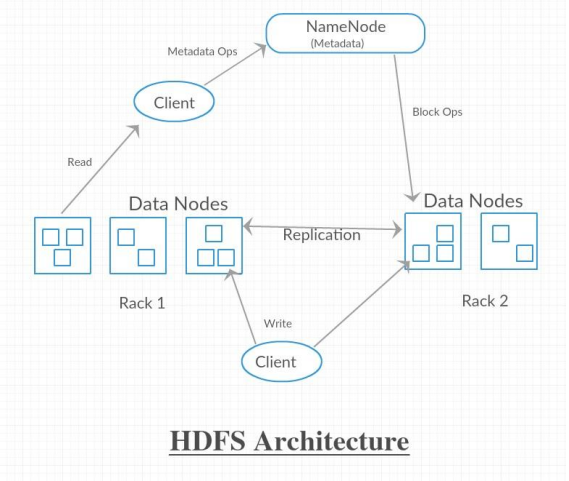

Figure: 2

Literature Survey

Applications frequently require more resources than are available on an inexpensive machine. Many organizations find themselves with business processes that no longer fit on a single cost effective computer. A simple but expensive solution has been to buy specialty machines that have a lot of memory and many CPUs. This solution scales as far as what is supported by the fastest machines available, and usually the only limiting factor is the budget. An alternative solution is to build a high-availability cluster. Such a cluster typically attempts to look like a single machine, and typically requires very specialized installation and administration services. Many high-availability clusters are proprietary and expensive. A more economical solution for acquiring the necessary computational resources is cloud computing. A common pattern is to have bulk data that needs to be transformed, where the processing of each data item is essentially independent of other data items; that is, using a single-instruction multiple-data (SIMD) algorithm. Apache Hadoop provides an open source framework for cloud computing, as well as a distributed file system. Apache Hadoop supports the MapReduce model, which was introduced by Google as a method of solving a class of petascale problems with large clusters of inexpensive machines. The model is based on two distinct steps for an application:

• **Map**: An initial ingestion and transformation step, in which individual input records can be processed in parallel.

• **Reduce:** An aggregation or summarization step, in which all associated records must be processed together by a single entity.

The core concept of MapReduce in Apache Hadoop is that input may be split into logical chunks and each chunk may be initially processed independently, by a map task. The results of these individual processing chunks can be physically partitioned into distinct sets, which are then sorted. Each sorted chunk is passed to a reduce task.
A map task may run on any compute node in the cluster, and multiple map tasks may be running in parallel across the cluster. The map task is responsible for transforming the input records into key/value pairs. The output of all of the maps will be partitioned, and each partition will be
sorted. There will be one partition for each reduce task. Each partition's sorted keys and the values associated with the keys are then processed by the reduce task. There may be multiple reduce tasks running in parallel on the cluster. The application developer needs to provide only four items to the Apache Hadoop framework: the class
that will read the input records and transform them into one key/value pair per record, a map method, a reduce method, and a class that will transform the key/value pairs that the reduce method outputs into output records.
The Apache Hadoop MapReduce framework requires a shared file system. This shared file system does not need to be a system-level file system, as

long as there is a distributed file system plug-in available to the framework. When HDFS is used as the shared file system, Apache Hadoop is able to take advantage of knowledge about which node hosts a physical copy of input data, and will attempt to schedule the task that is to read that data, to run on that machine.
The Apache Hadoop Distributed File System (HDFS) MapReduce environment provides the user with a sophisticated framework to manage the execution of map and reduce tasks across a cluster of machines. The user is required to tell the framework the following:

• Location(s) in the distributed file system of the job input
• Location(s) in the distributed file system for the job output
• Input format
• Output format
• Class containing the map function
• Optionally the class containing the reduce function
• JAR file(s) containing the map and reduce functions and any support classes.

If a job does not need a reduce function, the user does not need to specify a reducer class, and a reduce phase of the job will not be run. The framework will partition the input, and schedule and execute map tasks across the cluster. If requested, it will sort the results of the map task and execute the reduce task(s) with the map output. The final output will be moved to the output directory, and the job status will be reported to the user. MapReduce is oriented around key/value pairs. The framework will convert each record of input into a key/value pair, and each pair will be input to the map function once. The map output is a set of key/value pairs—nominally one pair that is the transformed input pair, but it is perfectly acceptable to output multiple pairs. The map output pairs are grouped and sorted by key. The reduce function is called one time for each key, in sort sequence, with the key and the set of values that share that key. The reduce method may output an arbitrary number of key/value pairs, which are written to the output files in the job output directory. If the reduce output keys are unchanged from the reduce input keys, the final output will be sorted. The framework provides two processes that handle the management of MapReduce jobs:

• **TaskTracker:** It manages the execution of individual map and reduce tasks on a compute node in the cluster.
• **JobTracker:** It accepts job submissions, provides job monitoring and control, and manages the distribution of tasks to the TaskTracker nodes.

Generally, there is one JobTracker process per cluster and one or more TaskTracker processes per node in the cluster. The JobTracker is a single point of failure, and the JobTracker will work around the failure of individual TaskTracker processes. The Apache Hadoop Distributed File System (HDFS) is a file system that is designed for use for MapReduce jobs that read input in large chunks of input, process it, and write potentially large chunks of output. HDFS does not handle random access particularly well. For reliability, file data is simply mirrored to multiple storage nodes. This is referred to as replication in the Apache Hadoop community. As long as at least one replica of a data chunk is available, the consumer of that data will not know of storage server failures. HDFS services are provided by two processes:

• NameNode: It handles management of the file system metadata, and provides management and control services.

• DataNode: It provides block storage and retrieval services. There will be one NameNode process in an HDFS file system, and this is a single point of failure. Apache Hadoop Core provides recovery and automatic backup of the NameNode. There will be multiple DataNode processes within the cluster, with typically one DataNode process per storage node in a cluster.

In this paper we are going to create an API which will fetch the data from archrivals which is known as dead data into the object storage which is known as alive system.

Conclusion

As more and more data are collected, the analysis of these data requires scalable, and high-performing tools to provide analysis and insight in a timely fashion. Cloud provides many solutions to solve Big-Data analytics problems. Most Big-Data architecture solutions utilize multiple-data tools to build a complete solution that meets business requirements in the most cost-optimized, performant, and resilient way possible. The result is a scalable Big-Data architecture that scales along with your business on the cloud global infrastructure.

References

[1]http://www.itproportal.com/2013/12/20/big-data-5-major-advantages-of-hadoop/
[2] **https://aws.amazon.com/elasticmapreduce/**
[3]http://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html
[4] http://www.tutorialspoint.com/sqoop/
[5] https://hadoop.apache.org/