UNIT II

A Brief History of Apache Hadoop

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine,itself a part of the Lucene project.

In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community. By this time, Hadoop was being used by many other companies besides Yahoo!, such as Last.fm, Facebook, and the New York Times. In one well-publicized feat, the New York Times used Amazon's EC2 compute cloud to crunch through 4 terabytes of scanned archives from the paper, converting them to PDFs for the Web. The processing took less than 24 hours to run using 100 machines, and the project probably wouldn't have been embarked upon without the combination ofAmazon's pay-by-the-hour model (which allowed the NYT to access a large number ofmachines for a short period) and Hadoop's easy-to-use parallel programming model.

In April 2008, Hadoop broke a world record to become the fastest system to sort an entire terabyte of data. Running on a 910-node cluster, Hadoop sorted 1 terabyte in 209 seconds (just under 3.5 minutes), beating the previous year's winner of 297 seconds. InNovember of the same year, Google reported that its MapReduce implementation sorted 1 terabyte in 68 seconds.Then, in April 2009, it was announced that a team at Yahoo! had used Hadoop to sort 1 terabyte in 62 seconds.

The trend since then has been to sort even larger volumes of data at ever faster rates. In the 2014 competition, a team from Databricks were joint winners of the Gray Sort benchmark. They used a 207-node Spark cluster to sort 100 terabytes of data in 1,406 seconds, a rate of 4.27 terabytes per minute.

Today, Hadoop is widely used in mainstream enterprises. Hadoop's role as a generalpurpose storage and analysis platform for big data has been recognized by the industry, and this fact is reflected in the number of products that use or incorporate Hadoop in some way. Commercial Hadoop support is available from large, established enterprise vendors, including EMC, IBM, Microsoft, and Oracle, as well as from specialist Hadoop companies such as Cloudera, Hortonworks, and MapR.

HADOOP BASICS:

- Need to process huge datasets on large clusters of computers
- Very expensive to build reliability into each application
- Nodes fail every day

Failure is expected, rather than exceptional
The number of nodes in a cluster is not constant

- Need a common infrastructure

Efficient, reliable, easy to use
Open Source, Apache Licence

Key Benefit & Flexibility

Client-server Concept
•Client sends requests to one or more servers which in turn accepts, processes them and return the requested information to the client.

•A server might run a software which listens on particular ipand port number for requests

•Examples:
Server -web server

Client –web browser

- Very Large Distributed File System

10K nodes, 100 million files, 10PB

- Assumes Commodity Hardware

Files are replicated to handle hardware failure
Detect failures and recover from them

- Optimized for Batch Processing

Data locations exposed so that computations can move to where data resides
Provides very high aggregate bandwidth

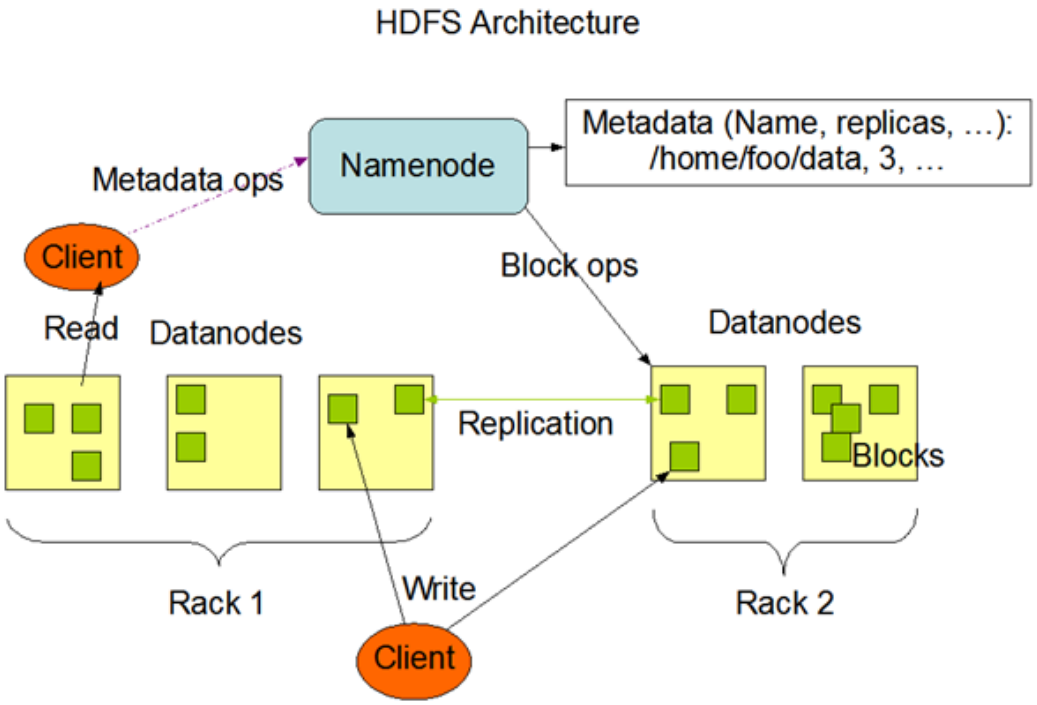
SATHYABAMA UNIVERSITY

COURSE MATERIAL - BIG DATA (SIT1606)

FACULTY OF COMPUTING

HDFS INFRASTRUCTURE AND ARCHITECTURE

Hadoop Distributed File System (HDFS)



Functions of a NameNode

- Manages File System Namespace
- Maps a file name to a set of blocks
- Maps a block to the DataNodes where it resides
- ClusterConfiguration Management
- Replication Engine for Blocks

NameNode Metadata

- Metadata in Memory

The entire metadata is in main memory
No demand paging of metadata

- Types of metadata

List of files
List of Blocks for each file
List of DataNodes for each block
File attributes, e.g. creation time, replication factor

- A Transaction Log

Records file creations, file deletions etc

DataNode

- A Block Server

Stores data in the local file system (e.g. ext3)
Stores metadata of a block (e.g. CRC)
Serves data and metadata to Clients

- Block Report

Periodically sends a report of all existing blocks to the NameNode

- Facilitates Pipelining of Data

Forwards data to other specified DataNodes

Block Placement

- Current Strategy

One replica on local node
Second replica on a remote rack
Third replica on same remote rack
Additional replicas are randomly placed

- Clients read from nearest replicas
- Would like tomake this policy pluggable

Heartbeats

- DataNodes send hearbeat to the NameNode
- Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

Replication Engine

- NameNode detects DataNode failures

Chooses new DataNodes for new replicas
Balances disk usage
Balances communication traffic to DataNodes

Data Correctness

- Use Checksums to validate data

Use CRC32

- File Creation

Client computes checksum per 512 bytes
DataNode stores the checksum

- File access

Client retrieves the data and checksum from DataNode
If Validation fails, Client tries other replicas

NameNode Failure

- A single point of failure
- Transaction Log stored in multiple directories

A directory on the local file system
A directory on a remote file system (NFS/CIFS)

- Need to develop a real HA solution

Data Pieplining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next node in the Pipeline
- When all replicas arewritten, the Client moves on to write the next block in file

Rebalancer

- Goal: % disk full on DataNodes should be similar

Usually run when new DataNodes are added
Cluster is online when Rebalancer is active

Rebalancer is throttled to avoid network congestion

Command line tool

Secondary NameNode

- Copies FsImage and Transaction Log from Namenode to a temporary directory
- Merges FSImage and Transaction Log into a new FSImage in temporary directory
- Uploads new FSImage to the NameNode

Transaction Log on NameNode is purged

User Interface

- Commads for HDFS User:

hadoopdfs -mkdir /foodir
hadoopdfs -cat /foodir/myfile.txt
hadoopdfs -rm /foodir/myfile.txt

- Commands for HDFS Administrator

hadoopdfsadmin -report
hadoopdfsadmin -decommisiondatanodename

- Web Interface

http://host:port/dfshealth.jsp

SATHYABAMA UNIVERSITY

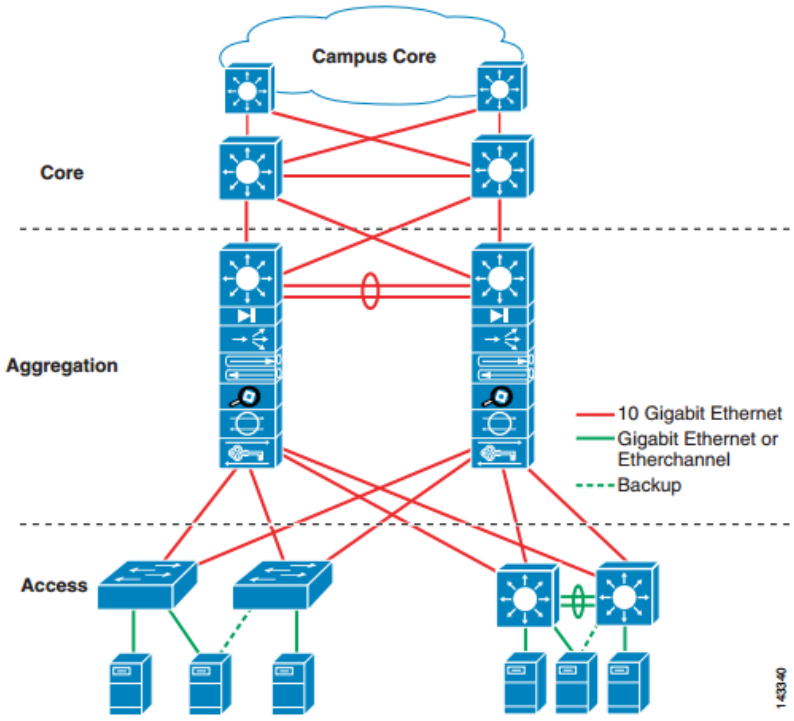
COURSE MATERIAL - BIG DATA (SIT1606)

FACULTY OF COMPUTING

Data Center Architecture

Overview:

The data center is home to the computational power, storage, and applications necessary to support an enterprise business. The data center infrastructure is central to the IT architecture, from which all content is sourced or passes through. Proper planning of the data center infrastructure design is critical, and performance, resiliency, and scalability need to be carefully considered. Another important aspect of the data center design is flexibility in quickly deploying and supporting new services. Designing a flexible architecture that has the ability to support new applications in a short time frame can result in a significant competitive advantage. Such a design requires solid initial planning and thoughtful consideration in the areas of port density, access layer uplink bandwidth, true server capacity, and oversubscription, to name just a few. The data center network design is based on a proven layered approach, which has been tested and improved over the past several years in some of the largest data center implementations in the world. The layered approach is the basic foundation of the data center design that seeks to improve scalability, performance, flexibility, resiliency, and maintenance.



The layers of the data center design are the core, aggregation, and access layers. These layers are referred to extensively throughout this guide and are briefly described as follows: • Core layer—Provides the high-speed packet switching backplane for all flows going in and out of the data center. The core layer provides connectivity to multiple aggregation modules and provides a resilient Layer 3 routed fabric with no single point of failure. The core layer runs an interior routing protocol, such as OSPF or EIGRP, and load balances traffic between the campus core and aggregation layers using Cisco Express Forwarding-based hashing algorithms. • Aggregation layer modules—Provide important functions, such as service module integration, Layer 2 domain definitions, spanning tree processing, and default gateway redundancy. Server-to-server multi-tier traffic flows through the aggregation layer and can use services, such as firewall and server load balancing, to optimize and secure applications. The smaller icons within the aggregation layer switch represent the integrated service modules. These modules provide services, such as content switching, firewall, SSL offload, intrusion detection, network analysis, and more. • Access layer—Where the servers physically attach to the network. The server components consist of 1RU servers, blade servers with integral switches, blade servers with pass-through cabling, clustered servers, and mainframes with OSA adapters. The access layer network infrastructure consists of modular switches, fixed configuration 1 or 2RU switches, and integral blade server switches. Switches provide both Layer 2 and Layer 3 topologies, fulfilling the various server broadcast domain or administrative requirements.

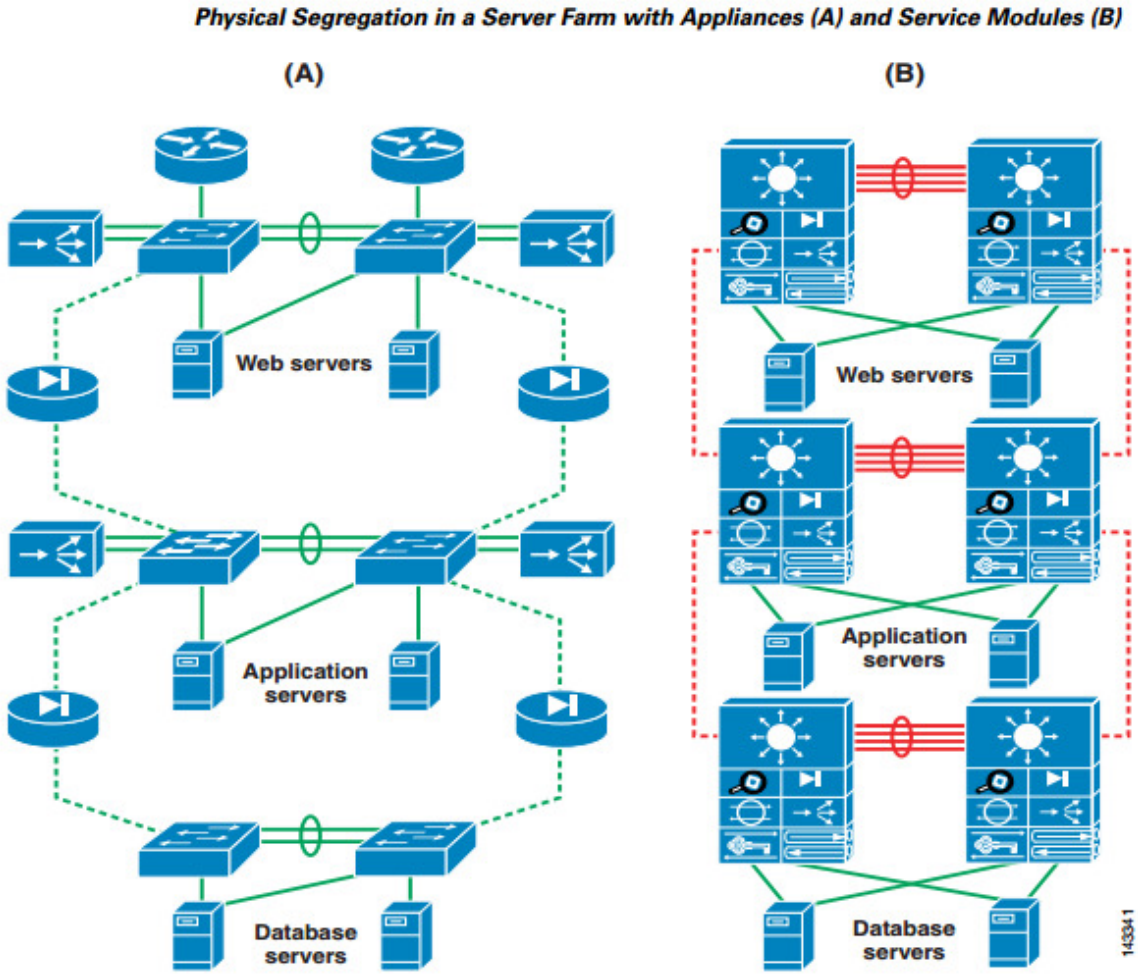
Data Center Design Models

Multi-Tier Model

The multi-tier data center model is dominated by HTTP-based applications in a multi-tier approach. The multi-tier approach includes web, application, and database tiers of servers. Today, most web-based applications are built as multi-tier applications. The multi-tier model uses software that runs as separate processes on the same machine using interprocess communication (IPC), or on different machines with communications over the network. Typically, the following three tiers are used: • Web-server • Application • Database Multi-tier server farms built with processes running on separate machines can provide improved resiliency and security. Resiliency is improved because a server can be taken out of service while the same function is still provided by another server belonging to the same application tier. Security is improved because an attacker can compromise a web server without gaining access to the application or database servers. Web and application servers can coexist on a common physical server; the database typically remains separate.

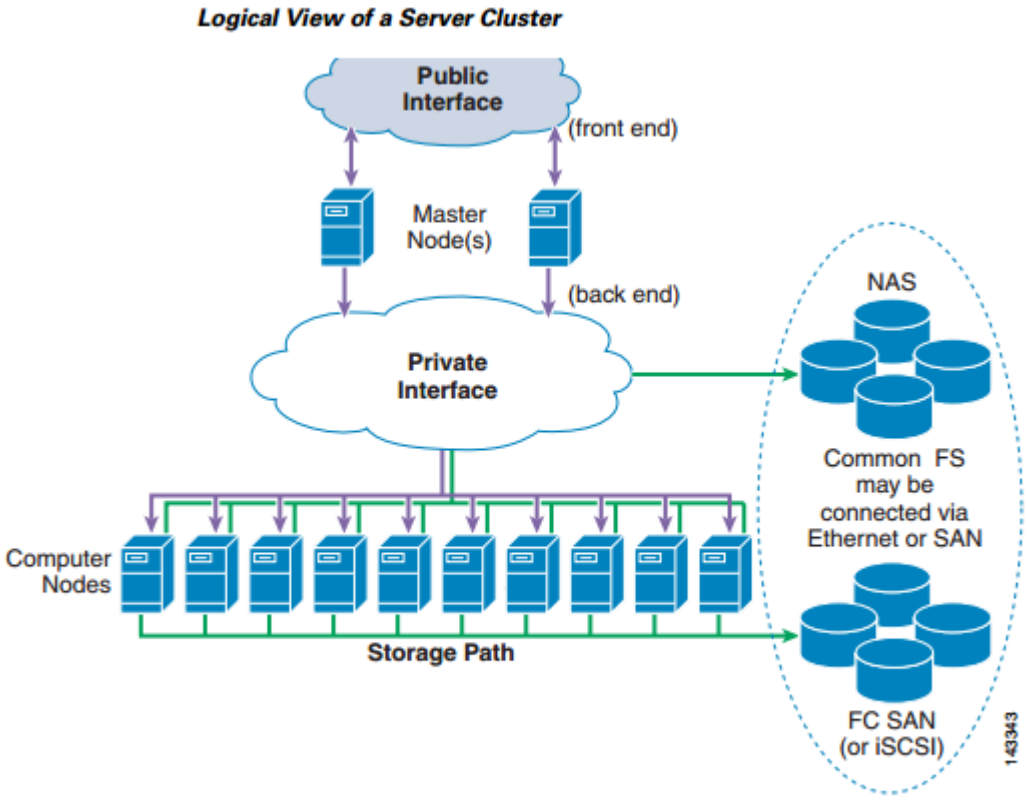
Resiliency is achieved by load balancing the network traffic between the tiers, and security is achieved by placing firewalls between the tiers. You can achieve segregation between the tiers

by deploying a separate infrastructure composed of aggregation and access switches, or by using VLANs



Server Cluster Model

In the modern data center environment, clusters of servers are used for many purposes, including high availability, load balancing, and increased computational power. This guide focuses on the high performance form of clusters, which includes many forms. All clusters have the common goal of combining multiple CPUs to appear as a unified high performance system using special software and high-speed network interconnects. Server clusters have historically been associated with university research, scientific laboratories, and military research for unique applications, such as the following: • Meteorology (weather simulation) • Seismology (seismic analysis) • Military research (weapons, warfare)



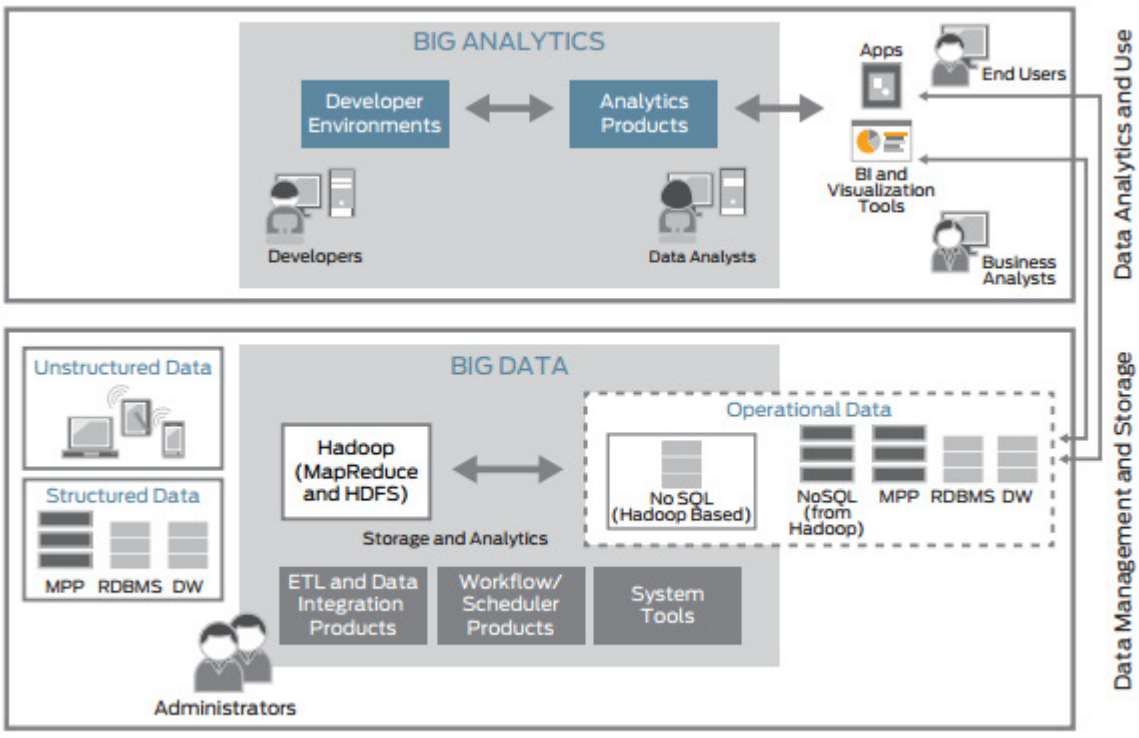
Logical Overview

The components of the server cluster are as follows: • Front end—These interfaces are used for external access to the cluster, which can be accessed by application servers or users that are submitting jobs or retrieving job results from the cluster. An example is an artist who is submitting a file for rendering or retrieving an already rendered result. This is typically an Ethernet IP interface connected into the access layer of the existing server farm infrastructure. • Master nodes (also known as head node)—The master nodes are responsible for managing the compute nodes in the cluster and optimizing the overall compute capacity. Usually, the master node is the only node that communicates with the outside world. Clustering middleware running on the master nodes provides the tools for resource management, job scheduling, and node state monitoring of the computer nodes in the cluster. Master nodes are typically deployed in a redundant fashion and are usually a higher performing server than the compute nodes. • Back-end high-speed fabric—This high-speed fabric is the primary medium for master node to compute node and inter-compute node communications. Typical requirements include low

latency and high bandwidth and can also include jumbo frame and 10 GigE support. Gigabit Ethernet is the most popular fabric technology in use today for server cluster implementations, but other technologies show promise, particularly Infiniband

• Compute nodes—The compute node runs an optimized or full OS kernel and is primarily responsible for CPU-intense operations such as number crunching, rendering, compiling, or other file manipulation. • Storage path—The storage path can use Ethernet or Fibre Channel interfaces. Fibre Channel interfaces consist of 1/2/4G interfaces and usually connect into a SAN switch such as a Cisco MDS platform. The back-end high-speed fabric and storage path can also be a common transport medium when IP over Ethernet is used to access storage. Typically, this is for NFS or iSCSI protocols to a NAS or SAN gateway, such as the IPS module on a Cisco MDS platform. • Common file system—The server cluster uses a common parallel file system that allows high performance access to all compute nodes. The file system types vary by operating system (for example, PVFS or Lustre)

HADOOP INFRASTRUCTURE AND ARCHITECTURE



High level structure of Hadoop deployment

Driven by a combination of technology innovations, maturing open source software, commodity hardware, ubiquitous social networking, and pervasive mobile devices, the rise of big data has created an inflection point making real-time data collection and analysis mission critical for businesses today. However, given that the data and its structures are fundamentally different, it is increasingly evident that the infrastructure, tools, and architectures to support real-time analysis and insight from this data also must be different. As an IT solution, big data mirrors the growth in both content and data source, as well as the pervasiveness of technology in our every day lives. As more and more of what we do is both connected to and often empowered by a network—and the devices that we connect to are themselves powered by an array of sensors—we should expect that the ongoing stream of data will grow. Within data centers, every node (servers, storage, and applications) generates a tremendous number of log files and isolated data streams that also can be collected, collated, and analyzed. With storage costs dropping, the cost associated with saving and leveraging even the most mundane data becomes a nonissue.

Understanding Data Traffic Flows

Within the last 20 years, data center infrastructure has been designed in a manner that closely aligns data, applications, and end users to provide secure, high-performance access. These silos have become rather commonplace, and network administrators can safely assume that the biggest consumer of an application and its corresponding data is an intelligent endpoint that can provide dedicated resources for compilation, execution, and display. This infrastructure has often been referred to as a three-tier architecture. The computing, storage, and networking to support this tiered architecture is largely optimized to deliver data and corresponding network traffic up and down the integrated stack to an end user and back to the database or storage (often referred to as north-south traffic). During the last few years, this predominant traffic pattern has changed dramatically. Big data represents just the latest application environment to drive this architectural shift. As data becomes more horizontally scaled and distributed throughout network nodes, traffic between server and storage nodes has become significantly greater than between servers and end users. Interestingly enough, the data itself can be generated by servers, applications, or storage environments as opposed to an external source (system log files, for example). This machine-to-machine network traffic and data sharing is often referred to as east-west traffic. Building a data center optimized to provide high-speed connections optimized for east-west traffic is critical in developing scalable, high-performance big data implementations.

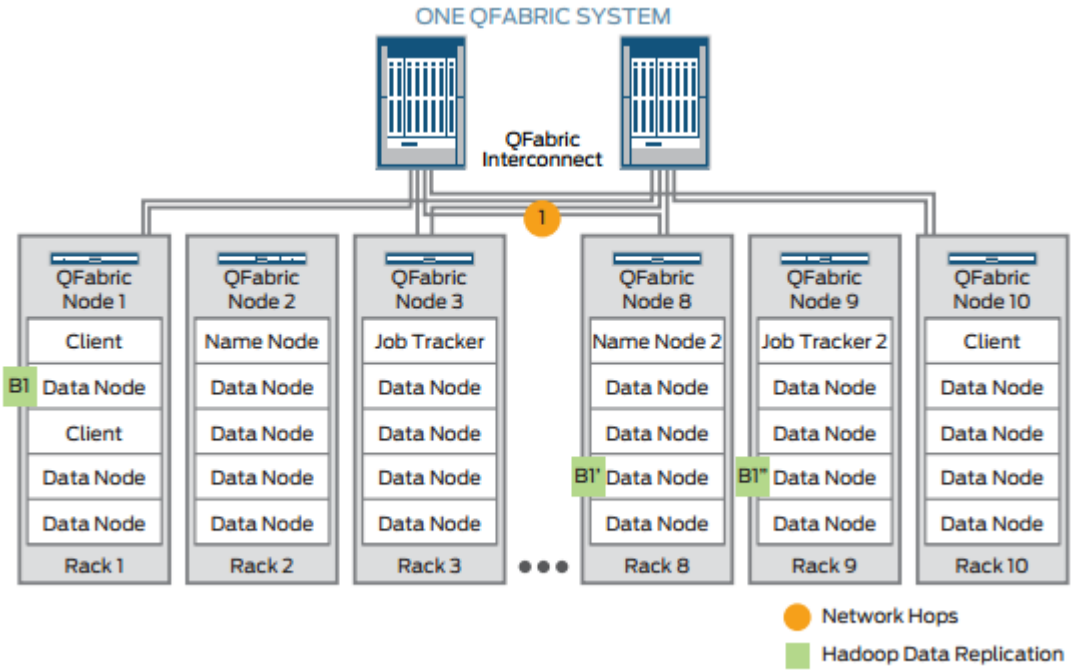
A Scalable Data Infrastructure :

Another unique characteristic of big data is that, unlike large data sets that have historically been stored and analyzed, often through data warehousing, big data is made up of discretely small, incremental data elements with real-time additions or modifications. It does not work well in traditional, online transaction processing (OLTP) data stores or with traditional SQL analysis tools. Big data requires a flat, horizontally scalable database, often with unique query tools that work in real time with actual data (as opposed to time delineated snapshots). Table 1 compares traditional data with big data.

The Hadoop Cluster

Hadoop, which includes a distributed file system known as Hadoop Distributed File System (HDFS) and MapReduce, is a critical big data technology that provides a scalable file system infrastructure and allows for the horizontal scale of data for quick query, access, and data management. At its most basic level, a Hadoop implementation creates four unique node types for cataloging, tracking, and managing data throughout the infrastructure: data node, client node, name node, and job tracker. For more information on Hadoop and related technologies along with implementation best practices, please see the Conclusion for additional resources. The capabilities of these four types are generally as follows: • Data node—The data nodes are the repositories for the data, and consist of multiple smaller database infrastructures that are horizontally scaled across compute and storage resources through the infrastructure. Larger big data repositories will have numerous data nodes. The critical architectural concern is that unlike traditional database infrastructure, these data nodes have no necessary requirement for locality to clients, analytics, or other business intelligence. • Client—The client represents the user interface to the big data implementation and query engine. The client could be a server or PC with a traditional user interface. • Name node—The name node is the equivalent of the address router for the big data implementation. This node maintains the index and location of every data node. • Job tracker—The job tracker represents the software job tracking mechanism to distribute and aggregate search queries across multiple nodes for ultimate client analysis. Within each data node, there may exist several tens of server or data storage elements and its own switching tier connecting each storage element with the overall Hadoop cluster. Big data infrastructure purposely breaks the data into horizontally scaled nodes, which naturally adds latencies across nodes. This is important as locality and network hops represent potential latencies in the architecture. For example, Figure 2 shows a granular representation of a more sophisticated Hadoop big data implementation that illustrates a typical architecture for the individual data nodes in a cluster. Not only is it particularly more resource and potentially management intensive

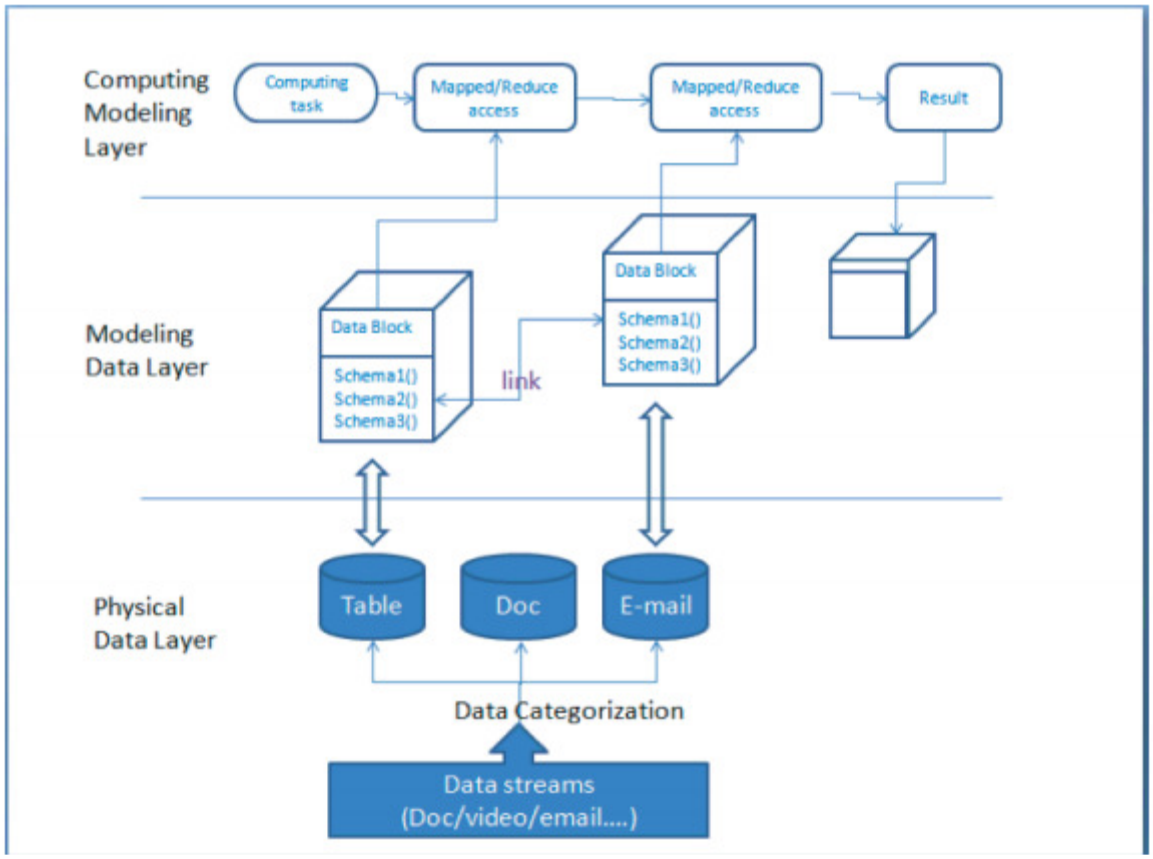
than a simple diagram may indicate, but from a pure performance perspective, the additional hops between client, job tracker, and individual data nodes are more significant. The job tracker is required to keep track of the five hops associated with top-of-rack switch 3 in order to access data with top-of-rack switch 8. These individual hops also represent latencies and potential performance bottlenecks.



Granular look at Hadoop node architecture

Data and Compute Modeling for Big Data

Big Data Model The big data model is an abstract layer used to manage the data stored in physical devices. Today we have large volumes of data with different formats stored in global devices. The big data model provides a visual way to manage data resources, and creates fundamental data architecture so that we can have more applications to optimize data reuse and reduce computing costs. The following diagram illustrates the general architecture of big data:



In the diagram above there are three model layers. The physical data layer is the data we have in a big data system. It can have different data types such as video, audio, logs, business tables, and so on. The data modeling layer is the abstract data model we build to manage physical data. The computing modeling layer is the application layer that we build to retrieve information for business value. With these three models, we build data models to separate physical data and data use. This means, the application is able to access data through the data model instead of accessing the physical data. This makes applications flexible and data manageable. To construct a big data model, we must first create data blocks based on data storage, data type, relationship, read-write requirement, and so on. Further, we must have modeling applications maintain these models, so that data models are able to display and store the latest data.

Hybrid Data Modeling

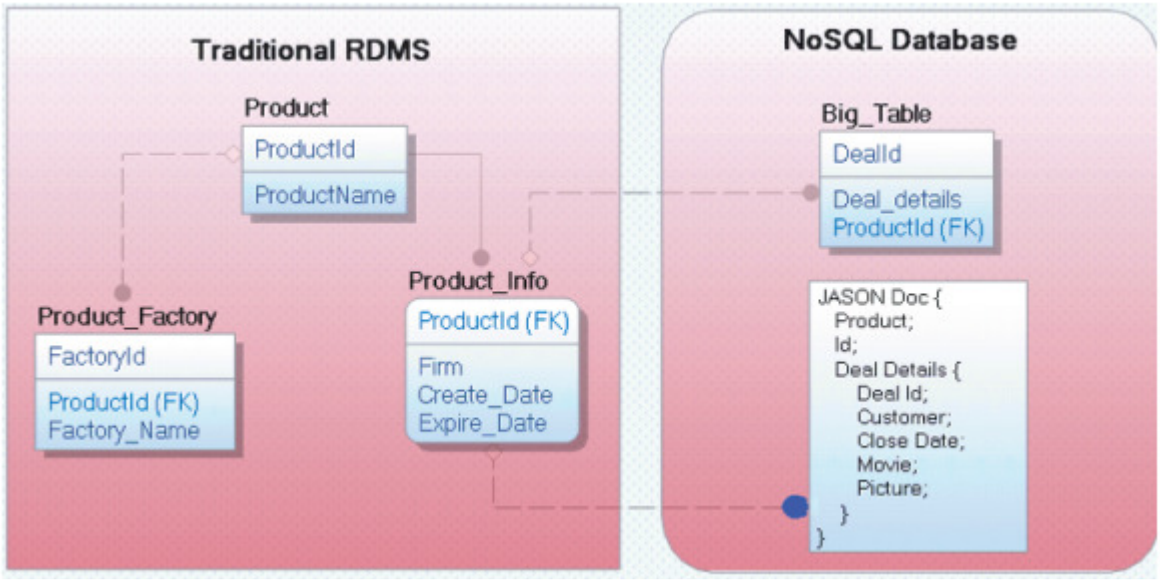
Though NoSQL databases evolved to resolve specific issues of managing big data, there are good reasons for SQL's enduring popularity, and current trends are to provide both SQL and

NoSQL features in the same database. Some traditional RDBMS vendors, such as Microsoft and Oracle, provide NoSQL features in recent releases, such as the columnar storage support in Microsoft SQL Server 2012 and Teradata Warehouse 14. From the NoSQL side, projects like the Hive subproject of Hadoop and the CQL query language of Cassandra, have grown quickly to provide an SQL-like façade to a NoSQL database. Google, the leader in big data, is also developing a hybrid distributed database, Megastore, that can run SQL-specific features with a NoSQL storage specification. That's not surprising - both the quantity and variety of data are growing rapidly and we need many models and tools to process it. In the past we tried to cram data of all types into an RDBMS; now we must keep some of it out. As Big Data drives changes in the collection and processing of data, we need to reconsider existing data and storage models to allow for the increasing importance of implementation issues such as the performance decrease imposed by join operations and pressures on storage space as data grows to exceed the capacity of hardware storage. There are new options to resolve these issues--we can elect to migrate big data into a NoSQL database, possibly running a hybrid system as a tradeoff, taking advantage of the low risks and costs of these promising new options.

Remodeling Tools and Techniques

In the age of big data, popular data modeling tools such as CA ERwin® Data Modeler continue to help us analyze and understand our data architectures by applying hybrid data modeling concepts. Instead of creating pure a relational data model, we now can embed NoSQLsubmodels within a relational data model. In general, data size and performance bottlenecks are the factors that help us decide which data goes to the NoSQL system. Furthermore, we can redesign models. For example, we can denormalize to reduce the dependency on relationships, and aggregate attributes from different entities into an entity in ER diagrams. Since we need to present the hierarchy of original data in the documentation, we need a good data representation format to assist in understanding original data relationship. The attribute-value orientation of JSON is well-suited to represent the key-value structure of many NoSQL stores, though XML still has its place. With this transformation in representation, modeling tools like CA ERwin Data Modeler can analyze schema changes and generate a code template to build a bridge between the current RDBMS and a target NoSQL system. This assists with data migration from an RDBMS to a NoSQL database, and gives the whole picture for data deployed in various data systems.

The following diagram illustrates data migration design:



Data Consumption

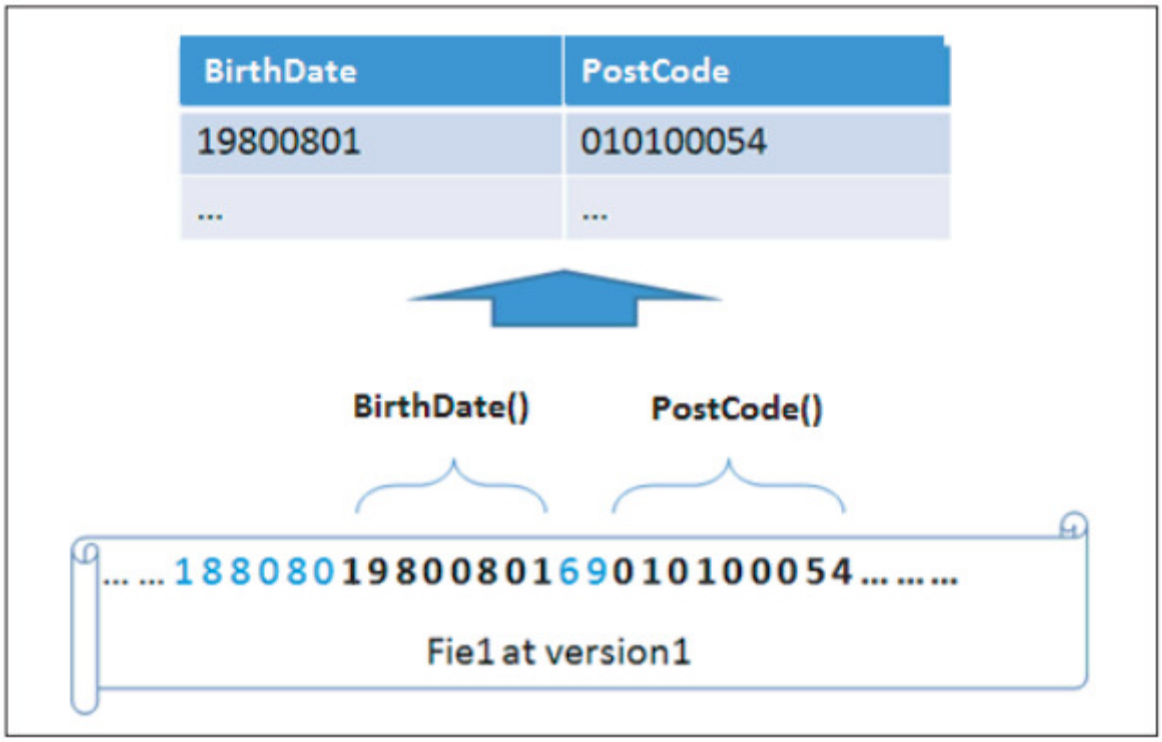
It often takes more effort to migrate data consumers to new storage in new system than to migrate the data itself. Once we have used a hybrid model to migrate data, applications designed to access the data using relational methodology must be redesigned to adapt to the new data architecture. Hopefully, future hybrid databases will provide built-in migration tools to help leverage the potential of NoSQL. In the present, remodeling data is critical before an existing system is migrated to NoSQL. In addition, modeling tools must to be evaluated to verify they meet the new requirements.

Physical Data-aware Modeling

For the past 30 years, 'structure first, collect later' has been the data modeling approach. With this approach, we determine the data structure before any data goes into the data system. In other words, the data structure definition is determined by how the data is used. With big data, this traditional approach is no longer applicable, as data doesn't necessarily have fixed schemas or formats. As a result, we need a new modeling methodology to suit big data characteristics and deployment. I propose a new approach for managing the variety of big data.

Dynamic Schema Definition:

Even with big unstructured or semi-structured data, we still need to define schemas because data relationships can be more complex than before. Hiding data relationship logic in a program is not a good way to manage data complexity. Because big data uses the 'structure later' approach, in most of the cases, we can only know the data schema after the data has been created. The proposed methodology is to define schema on existing data after it has been collected and stored, and use the schema to retrieve data at runtime while processing. In the definition, the schema function should be as atomic and isolated as possible. To achieve this, we must find out the scope of the data the schema function applies to, and the versions of data that the schema function can work for. This is called "dynamic schema" to distinguish it from traditional fixed schema defined before data is collected. The following is an example of a schema function showing how to fetch BirthDate and PostCode from an identity number



Data Block

A data block is the physical data defined with metadata. We must know the actual location of data, so we can model it on a modeling layer, and so the computing program knows where to retrieve it. Include the following fields in the metadata definition:

• Region servers, the physical location of data storage

• Data access path that the program can use to access data

• Data format of the saved data

• Other attributes added as needed when the data is used As a general rule to manage similar types of data, place data of identical formatand schema functions in the same data block.

The data in a data block should be stored in close physical location. For example, data that might be used at the same time should be stored in the same rack, or at least in the same region.

Data Version

In a real-time production scenario, data may have different formats at different times in its lifecycle. Data models should manage these versions of data, so that schema functions can ensure compatible data update. Schema functions should be version-aware, so historic data can be retrieved with the appropriate legacy schema function.

Data Relationship

The data relationship among multiple data blocks still exists in practice, although most of the data in the big data system are designed as relationshipless (that is, without joins) to facilitate scalability. However, denormalization may lead to data duplication in storage, and to problems in a distributed system if data has to be updated frequently. We need to balance between duplicating data for scalability and query performance and practical requirements for update performance. Relationships are defined on schema functions in data blocks. The relationships are just data links. Optional proxy applications can exist to maintain consistency for data changes, such as deleting data blocks or changing data format. The important benefit is computing tasks can fetch data using the data relationship definition in a bunch of data blocks. This also can be proactively designed in a modeling layer

Data Computing Modeling

On the top of the physical data model, we normally need to create data flow and compute tasks for business requirements. With physical-data modeling, it’s possible to create a computing model, which can present the logic path of computing data. This will help computing tasks be

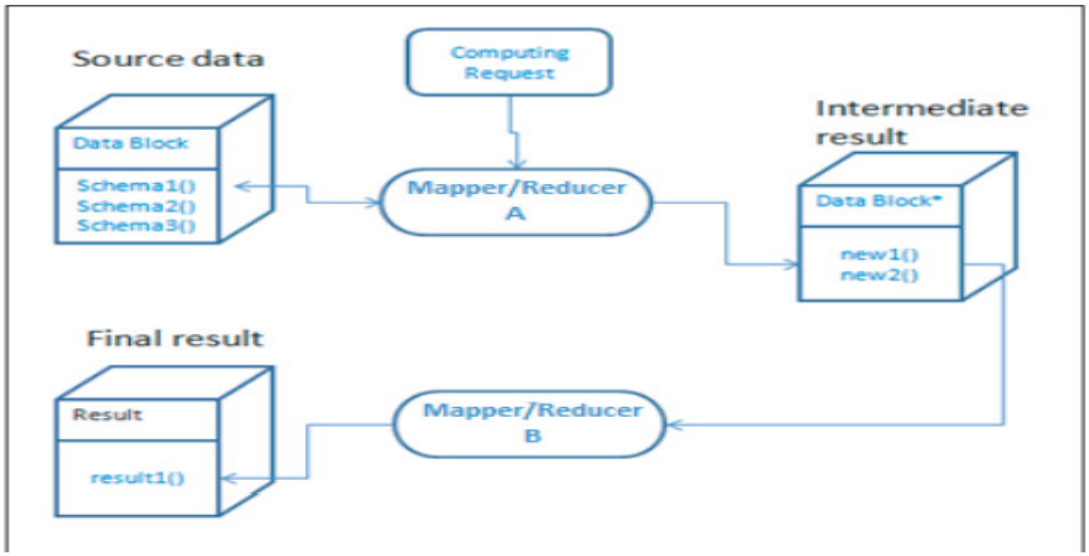
well designed and enable more efficient data reuse. Hadoop provides a new distributed data processing model, and its HBase database provides an impressive solution for data replication, backup, scalability, and so on. Hadoop also provides the Map/Reduce computing framework to retrieve value from data stored in a distributed system. Map/Reduce is a framework for parallel processing using mappers dividing a problem into smaller sub-problems to feed reducers that process the subproblems and produce the final answer. As a business grows, computing requirements becomes more and more complex. For example, some normal computing tasks might need a couple of Map/Reduce subtasks to meet specific business requirements. We need a better design on the dataflow of complex computing. Reuse of a existing Mapper/Reducer is also a potential requirement for effective production environments.

Computing Unit

A Mapper and Reducer pair is a minimum unit to compute the final output based on the input. A good design is one that is simple, because it can aid debugging and provide a more stable environment. The key point is how to make the Mapper and Reducer pair more independent of physical data, so that we can better reuse these computing units to accelerate development and reduce cost.

Computing Flow

As the following diagram shows, rather than writing bigger Mappers and Reducer, we can combine existing Mappers and Reducers in a workflow. The output of the first Map-Reduce is the input of the second Map-Reduce. The whole work flow can be visualized and managed based on the data model introduced above



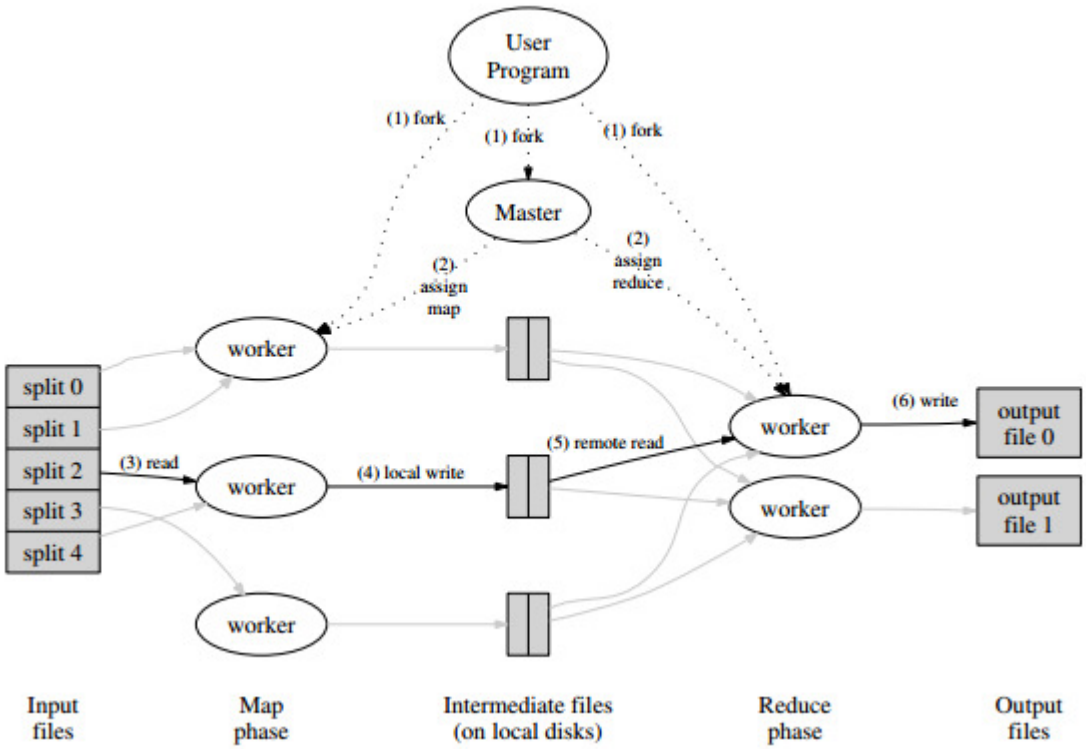
Data computing modeling helps manage the computing algorithm on the data stored in raw format. When new tools are developed, the data model will assist in taking the computing task forward, and help monitor the progress of data processing. Meanwhile, we can embed the data model into a big data system like Hadoop. Once data is moved or imported, Hadoop can synchronize with the data model first.

Integration with Hive Hive creates separate data systems for better querying and computing. The tables in Hive have well-defined schemas. We can reverse engineer to import schemas to specific data blocks and integrate Hive data and external RAW data in Hadoop'sHBase database in the same data model. This provides a useful basic architecture if Hive has to extract data from RAW data, and run computing tasks on both data sources during data mining.

The complexity of a relational database limits the scalability of data storage, but makes it very easy to query data through an SQL engine. Big data NoSQL systems have the opposite characteristics: unlimited scalability with more limited query capabilities. The challenge of big data is querying data easily. Creating data models on physical data and computing path help manage raw data. he future will bring more hybrid systems combining the attributes of both approaches. Meanwhile, the dynamic schema model proposed in this paper and systems such as Hive offer help in the challenge of managing big data.

MapreduceFrameWork:

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.



The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The input splits can be processed in parallel by different machines. Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e.g., hash(key) mod R). The number of partitions (R) and the partitioning function are specified by the user. Figure 1 shows the overall flow of a MapReduce operation in our implementation. When the user program calls the MapReduce function, the following sequence of actions occurs (the numbered labels in Figure 1 correspond to the numbers in the list below): 1. The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece (controllable by the user via an optional parameter). It then starts up many copies of the program on a cluster of machines. 2. One of the copies of the program is special – the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task. 3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory. 4. Periodically, the

buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers. 5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. The sorting is needed because typically many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used. 6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this reduce partition.

7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code. After successful completion, the output of the mapreduce execution is available in the R output files (one per reduce task, with file names as specified by the user). Typically, users do not need to combine these R output files into one file – they often pass these files as input to another MapReduce call, or use them from another distributed application that is able to deal with input that is partitioned into multiple files.

Hadoop Operation Modes

Once you have downloaded Hadoop, you can operate your Hadoop cluster in one of the three supported modes:

- **Local/Standalone Mode** : After downloading Hadoop in your system, by default, it is configured in a standalone mode and can be run as a single java process.

- **Pseudo Distributed Mode** : It is a distributed simulation on single machine. Each Hadoop daemon such as hdfs, yarn, MapReduce etc., will run as a separate java process. This mode is useful for development.

- **Fully Distributed Mode** : This mode is fully distributed with minimum two or more machines as a cluster. We will come across this mode in detail in the coming chapters.

Installing Hadoop in Standalone Mode

Here we will discuss the installation of **Hadoop 2.4.1** in standalone mode.

There are no daemons running and everything runs in a single JVM. Standalone mode is suitable for running MapReduce programs during development, since it is easy to test and debug them.

Setting UpHadoop

You can set Hadoop environment variables by appending the following commands to **~/.bashrc** file.

```
export HADOOP_HOME=/usr/local/hadoop
```

Before proceeding further, you need to make sure that Hadoop is working fine. Just issue the following command:

```
$ hadoop version
```

If everything is fine with your setup, then you should see the following result:

```
Hadoop2.4.1
Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiledbyhortonmu on 2013-10-07T06:28Z
Compiledwithprotoc2.5.0
From source with checksum 79e53ce7994d1628b240f09af91e1af4
```

It means your Hadoop's standalone mode setup is working fine. By default, Hadoop is configured to run in a non-distributed mode on a single machine.

Example

Let's check a simple example of Hadoop. Hadoop installation delivers the following example MapReduce jar file, which provides basic functionality of

MapReduce and can be used for calculating, like Pi value, word counts in a given list of files, etc.

```
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar
```

Let's have an input directory where we will push a few files and our requirement is to count the total number of words in those files. To calculate the total number of words, we do not need to write our MapReduce, provided the .jar file contains the implementation for word count. You can try other examples using the same .jar file; just issue the following commands to check supported MapReduce functional programs by hadoop-mapreduce-examples-2.2.0.jar file.

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jar
```

Step 1

Create temporary content files in the input directory. You can create this input directory anywhere you would like to work.

```
$ mkdir input

$ cp $HADOOP_HOME/*.txt input

$ ls -l input
```

It will give the following files in your input directory:

```
total24

-rw-r--r--1 root root15164Feb2110:14 LICENSE.txt

-rw-r--r--1 root root101Feb2110:14 NOTICE.txt

-rw-r--r--1 root root1366Feb2110:14 README.txt
```

These files have been copied from the Hadoop installation home directory. For your experiment, you can have different and large sets of files.

Step 2

Let's start the Hadoop process to count the total number of words in all the files available in the input directory, as follows:

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jarwordcount
input output
```

Step 3

Step-2 will do the required processing and save the output in output/part-r00000 file, which you can check by using:

```
$cat output/*
```

It will list down all the words along with their total counts available in all the files available in the input directory.

```
"AS      4
"Contribution" 1
"Contributor" 1
"Derivative1
"Legal 1
"License"      1
"License");     1
"Licensor"      1
"NOTICE”1
"Not      1
"Object"        1
"Source”1
"Work”    1
"You"     1
"Your")   1
"[]"      1
```

```
"control"       1
"printed1
"submitted"1
(50%)1
(BIS),1
(C)1
(Don't)   1
(ECCN)    1
(INCLUDING      2
(INCLUDING,     2
.............
```

Installing Hadoop in Pseudo Distributed Mode

Installing Hadoop in Pseudo Distributed Mode

Follow the steps given below to install Hadoop 2.4.1 in pseudo distributed mode.

Step 1: Setting Up Hadoop

You can set Hadoop environment variables by appending the following commands to **~/.bashrc** file.

```
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

```
export HADOOP_INSTALL=$HADOOP_HOME
```

Now apply all the changes into the current running system.

```
$ source~/.bashrc
```

Step 2: Hadoop Configuration

You can find all the Hadoop configuration files in the location "$HADOOP_HOME/etc/hadoop". It is required to make changes in those configuration files according to your Hadoop infrastructure.

```
$ cd $HADOOP_HOME/etc/hadoop
```

In order to develop Hadoop programs in java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing **JAVA_HOME**value with the location of java in your system.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
```

The following are the list of files that you have to edit to configure Hadoop.

core-site.xml

The **core-site.xml** file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and size of Read/Write buffers.

Open the core-site.xml and add the following properties in between <configuration>, </configuration> tags.

```
<configuration>


<property>
<name>fs.default.name </name>
<value> hdfs://localhost:9000 </value>
```

```
</property>


</configuration>
```

hdfs-site.xml

The **hdfs-site.xml** file contains information such as the value of replication data, namenode path, and datanode paths of your local file systems. It means the place where you want to store the Hadoop infrastructure.

Let us assume the following data.

```
dfs.replication(data replication value)=1
(In the below given path /hadoop/is the user name.
hadoopinfra/hdfs/namenodeis the directory created byhdfs file system.)
namenode path =//home/hadoop/hadoopinfra/hdfs/namenode
(hadoopinfra/hdfs/datanodeis the directory created byhdfs file system.)
datanode path =//home/hadoop/hadoopinfra/hdfs/datanode
```

Open this file and add the following properties in between the <configuration></configuration> tags in this file.

```
<configuration>


<property>
<name>dfs.replication</name>
<value>1</value>
</property>


<property>
<name>dfs.name.dir</name>
<value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>
```

```
</property>


<property>
<name>dfs.data.dir</name>
<value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value>
</property>


</configuration>
```

Note: In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.

yarn-site.xml

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>


<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>


</configuration>
```

mapred-site.xml

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, it is required to copy the file from **mapred-site,xml.template** to **mapred-site.xml** file using the following command.

```
$ cpmapred-site.xml.template mapred-site.xml
```

Open mapred-site.xml file and add the following properties in between the <configuration>, </configuration>tags in this file.

```
<configuration>


<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>


</configuration>
```

Verifying Hadoop Installation

The following steps are used to verify the Hadoop installation.

Step 1: Name Node Setup

Set up the namenode using the command "hdfsnamenode -format" as follows.

```
$ cd~
$ hdfsnamenode-format
```

The expected result is as follows.

```
10/24/1421:30:55 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = localhost/192.168.1.11
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.4.1
```

```
...
...
10/24/14 21:30:56 INFO common.Storage: Storage directory
/home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.
10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to
retain 1 images with txid>= 0
10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0
10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at localhost/192.168.1.11
************************************************************/
```

Step 2: Verifying Hadoopdfs

The following command is used to start dfs. Executing this command will start your Hadoop file system.

```
$ start-dfs.sh
```

The expected output is as follows:

```
10/24/1421:37:56
Startingnamenodes on [localhost]
localhost: starting namenode, logging to /home/hadoop/hadoop
2.4.1/logs/hadoop-hadoop-namenode-localhost.out
localhost: starting datanode, logging to /home/hadoop/hadoop
2.4.1/logs/hadoop-hadoop-datanode-localhost.out
Starting secondary namenodes[0.0.0.0]
```

Step 3: Verifying Yarn Script

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

```
$ start-yarn.sh
```

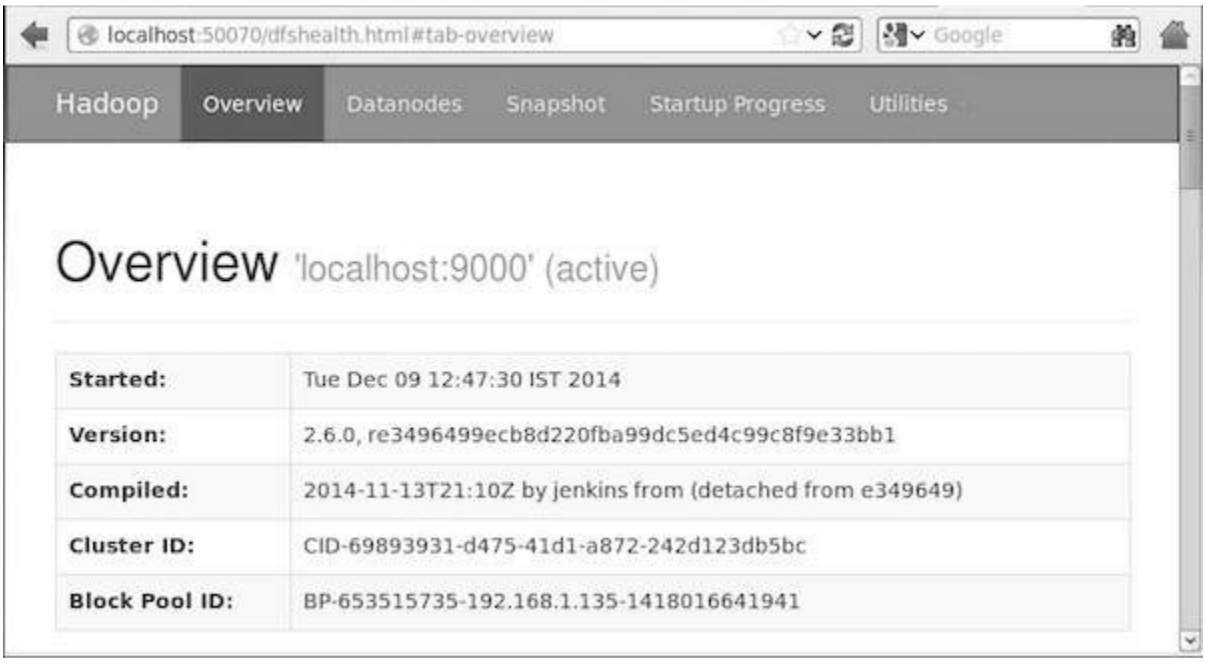
The expected output as follows:

```
starting yarn daemons
startingresourcemanager, logging to /home/hadoop/hadoop
2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out
localhost: starting nodemanager, logging to /home/hadoop/hadoop
2.4.1/logs/yarn-hadoop-nodemanager-localhost.out
```

Step 4: Accessing Hadoop on Browser

The default port number to access Hadoop is 50070. Use the following url to get Hadoop services on browser.

```
http://localhost:50070/
```



Step 5: Verify All Applications for Cluster

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

SATHYABAMA UNIVERSITY

COURSE MATERIAL - BIG DATA (SIT1606)

FACULTY OF COMPUTING

```
http://localhost:8088/
```