Design of HDFS

- Hadoop was created by Doug Cutting who had created the Apache Lucene (Text Search),which is origin in Apache Nutch (Open source search Engine).Hadoop is a part of Apache Lucene Project.Actually Apache Nutch was started in 2002 for working crawler and search
- In January 2008, Hadoop was made its own top-level project at Apache for, confirming success ,By this time, Hadoop was being used by many other companies such as Yahoo!, Facebook, etc.
- In April 2008, Hadoop broke a world record to become the fastest system to sort a terabyte of data.
- Yahoo take test in which To process 1TB of data (1024 columns) oracle

– 3 ½ day

teradata – 4 ½ day

netezza – 2 hour 50 min

hadoop - 3.4 min

Design of HDFS

WHAT IS HADOOP

- Hadoop is the product of Apach ,it is the type of distributed system, it is framework for big data
- Apache Hadoop is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware.
- Some of the characteristics:
  - Open source
  - Distributed processing
  - Distributed storage
  - Reliable
  - Economical
  - Flexible

Design of HDFS

Hadoop Framework Modules

- The base Apache Hadoop framework is composed of the following modules:
- **Hadoop Common :–** contains libraries and utilities needed by other Hadoop modules
- **Hadoop Distributed File System (HDFS) :–** a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster
- **Hadoop YARN:–** a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications
- **Hadoop MapReduce:–** an implementation of the MapReduce programming model for large scale data processing.

Design of HDFS

Basic Features: HDFS

- Highly fault-tolerant
- High throughput
- Suitable for applications with large data sets
- Streaming access to file system data
- Can be built out of commodity hardware

Design of HDFS

Fault tolerance

- Failure is the norm rather than exception
- A HDFS instance may consist of thousands of server machines, each storing part of the file system’s data.
- Since we have huge number of components and that each component has non-trivial probability of failure means that there is always some component that is non-functional.
- Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

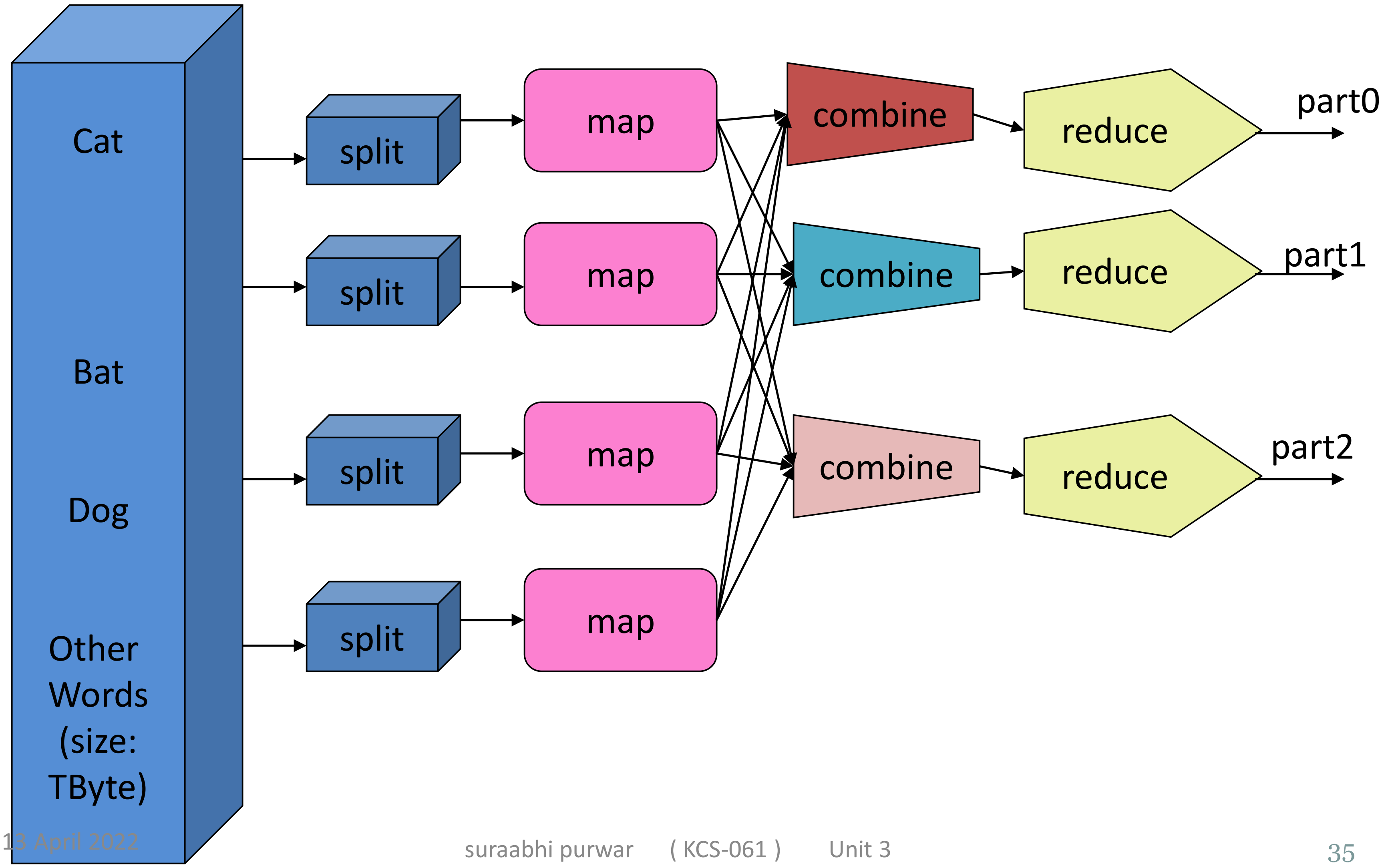
Design of HDFS

Data Characteristics

- Streaming data access
- Applications need streaming access to data
- Batch processing rather than interactive user access.
- Large data sets and files: gigabytes to terabytes size
- High aggregate data bandwidth
- Scale to hundreds of nodes in a cluster
- Tens of millions of files in a single instance
- Write-once-read-many: a file once created, written and closed need not be changed – this assumption simplifies coherency
- A map-reduce application or web-crawler application fits perfectly with this model.

Design of HDFS

MapReduce



HDFS concepts

Objective:

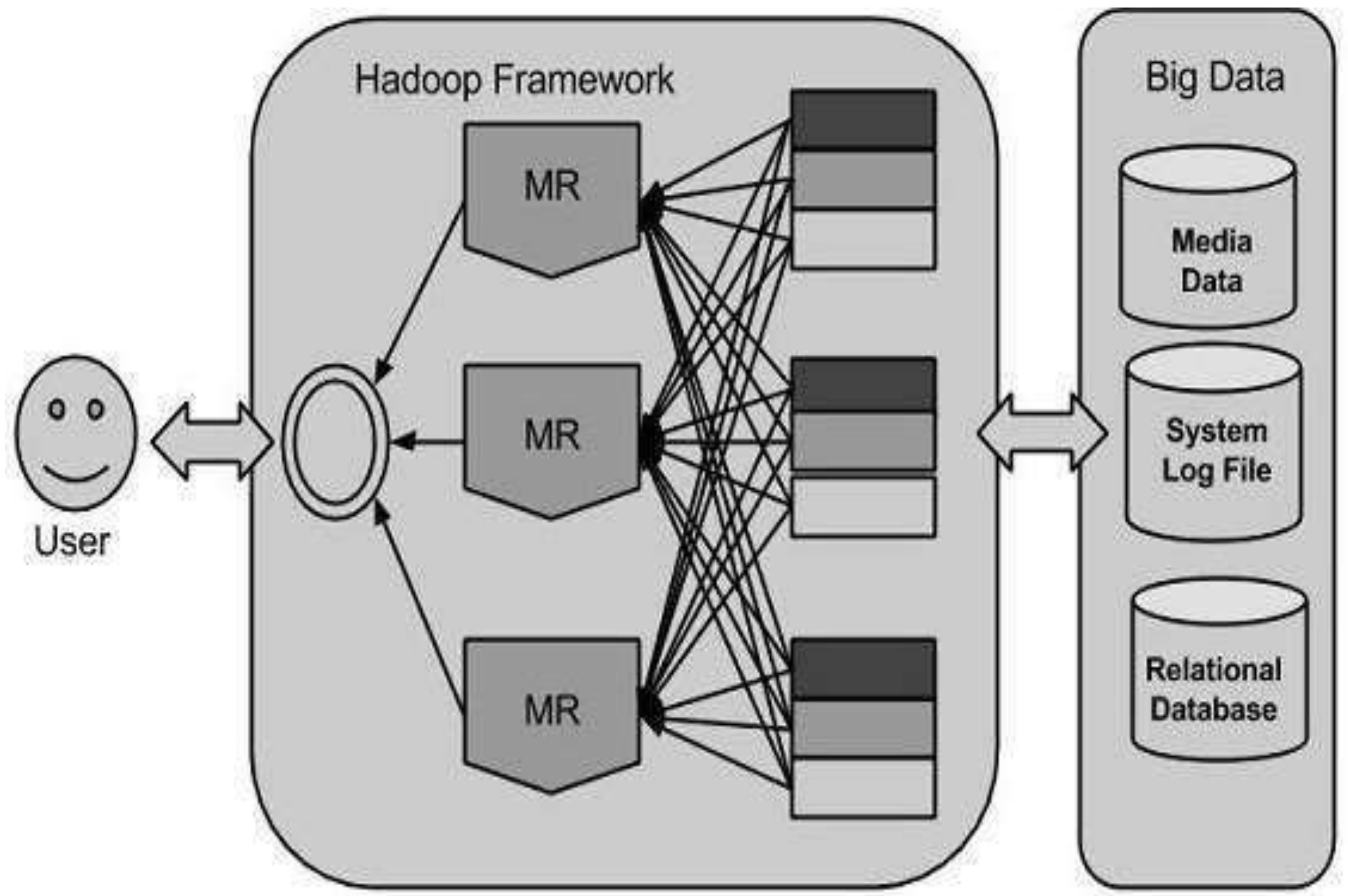
- **In this topic** we focus on Hadoop File System was developed using distributed file system design. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware. HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines.

Recap:

- Revision of HDFS Architecture.

HDFS concepts

Framework Architecture



HDFS concepts

Hadoop Services

- Storage
  1. HDFS (Hadoop distributed file System)
     a)Horizontally Unlimited Scalability(No Limit For Max no.of Slaves)
     b)Block Size=64MB(old Version)
     128MB(New Version)
- Process
  1. MapReduce(Old Model)
  2. Spark(New Model)

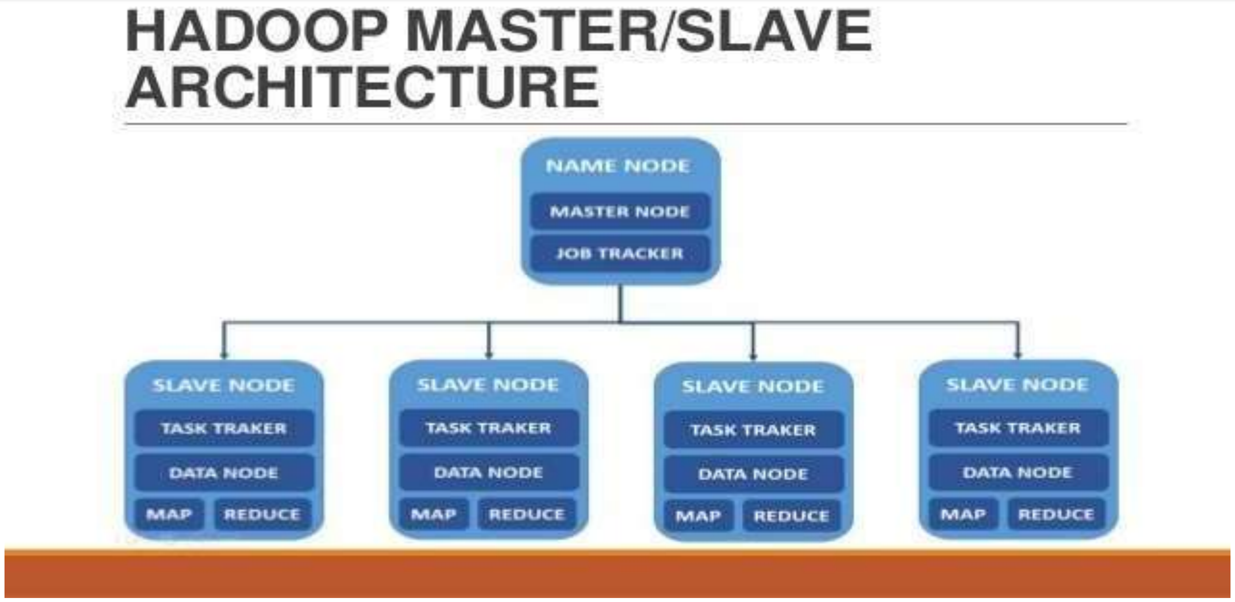
HDFS concepts

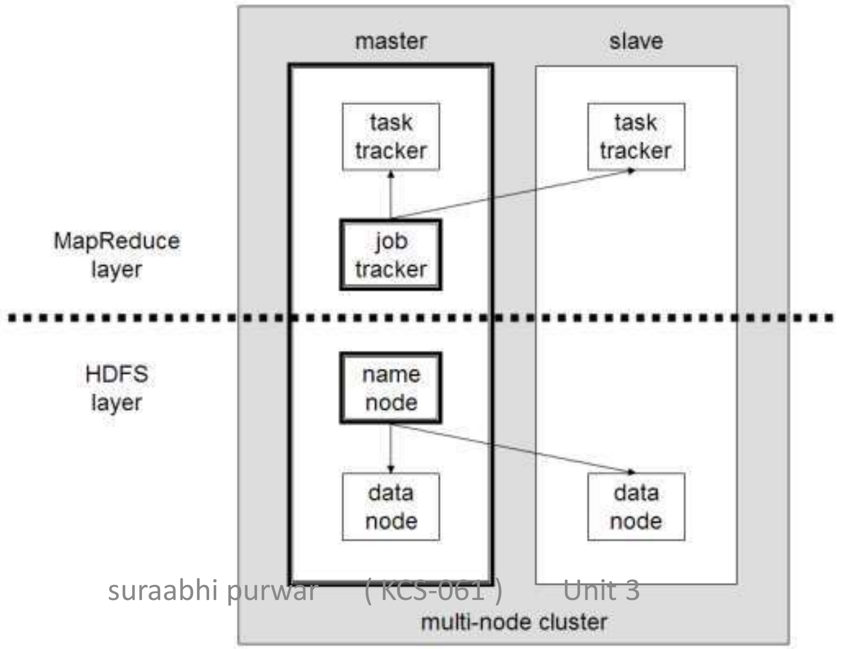
HadoopArchitecture

Hadoop consists of the Hadoop Common package, which provides file system and OS level abstractions, a

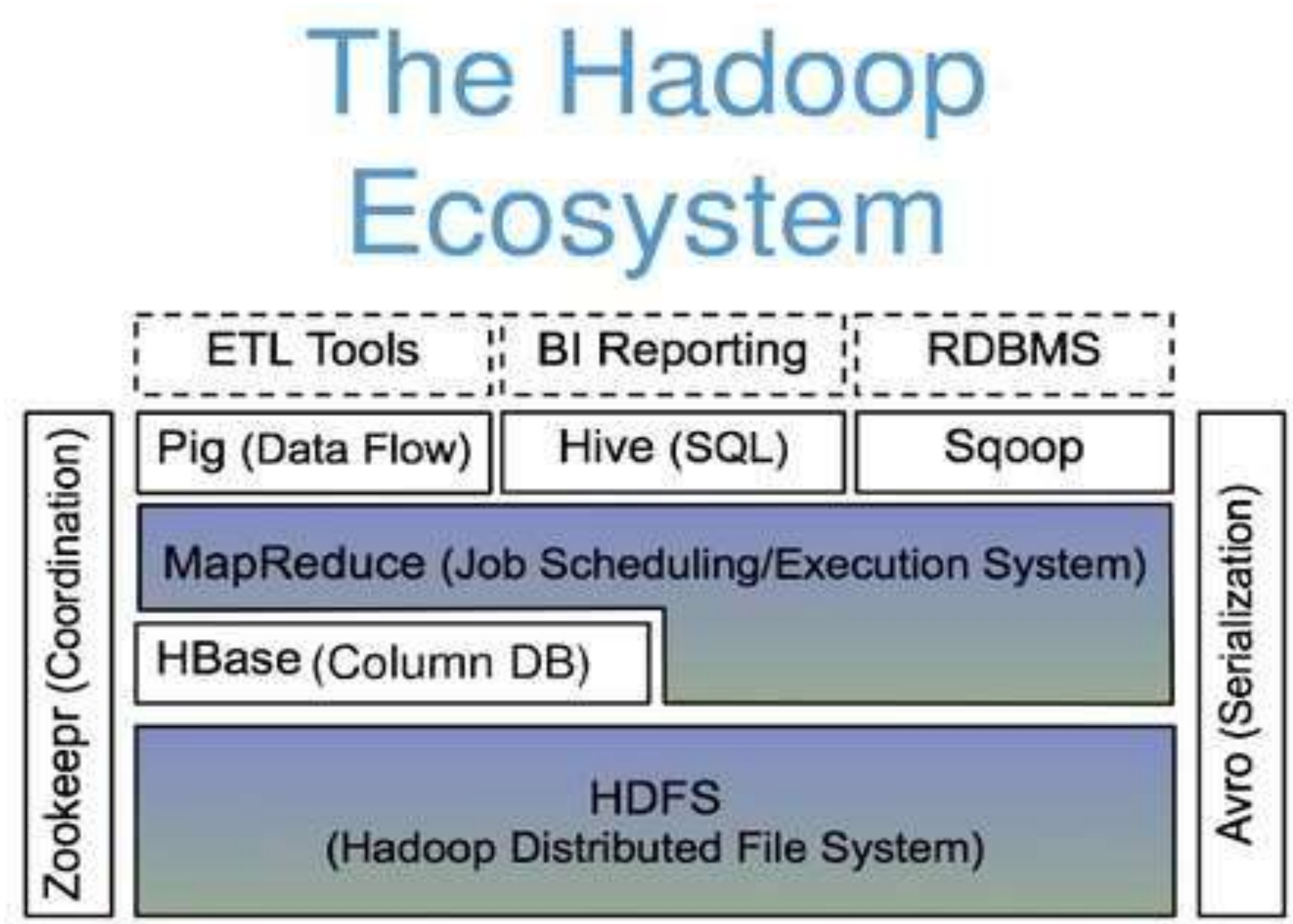
MapReduce engine and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the necessary Java Archive (JAR) files and scripts needed to start Hadoop.

HDFS concepts





HDFS concepts



Hadoop file system interfaces

Objective:

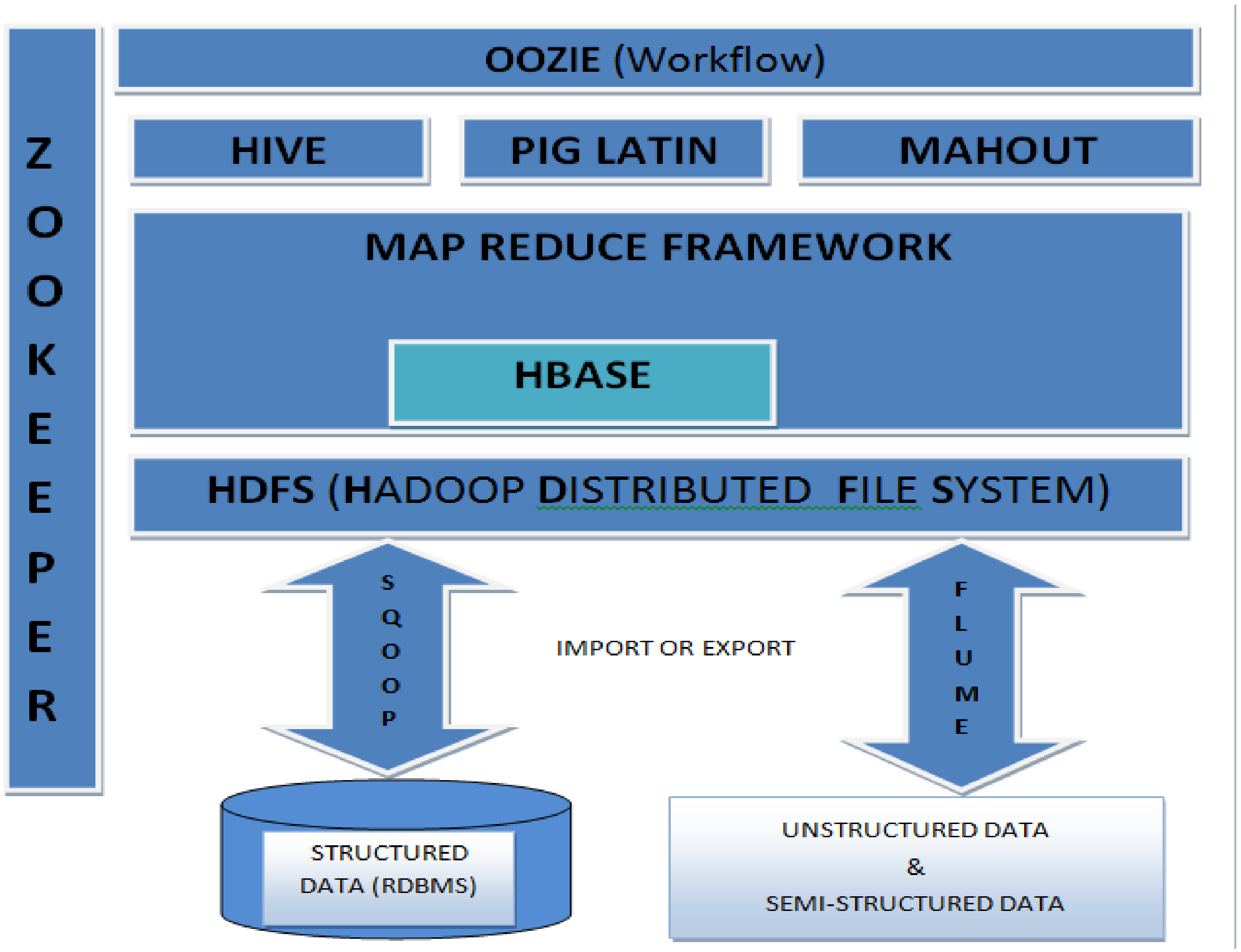
- **In this topic** we focus on A distributed file system offers a logical file system that is implemented over a cluster of machines rather than on a single machine. Additional details on HDFS are available on the Apache Hadoop website. Hadoop offers access to HDFS through a command line interface.

Recap:

- Revision of database systems.

Hadoop file system interfaces

Working Of Ecosystem



File sizes, block sizes and block abstraction in HDFS

Objective:

- **In this topic** we focus on HDFS Block abstraction: HDFS block size is of 64MB-128MB(usually) and unlike other filesystems, a file smaller than the block size does not occupy the complete block size's worth of memory. The block size is kept so large so that less time is made doing disk seeks as compared to the data transfer rate.

Recap:

- Revision of Big Data Framework.

File sizes, block sizes and block abstraction in HDFS

HDFS

- Hadoop Distributed File System (HDFS) is designed to reliably store very large files across machines in a large cluster. It is inspired by the GoogleFileSystem.
- Distribute large data file into blocks
- Blocks are managed by different nodes in the cluster
- Each block is replicated on multiple nodes
- Name node stored metadata information about files and blocks

File sizes, block sizes and block abstraction in HDFS

MAPREDUCE

- **The Mapper:-**

  1. Each block is processed in isolation by a map task called mapper
  2. Map task runs on the node where the block is stored

- **The Reducer:-**

  1. Consolidate result from different mappers
  2. Produce final output

THE CONCEPT LEARNING TASK

Objective:

- **In this topic** we focus on Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers.

Recap:

- Revision of database systems.

HBASE

- Hadoop database for random read/write access
- **Features of HBASE:-**
  1. Type of NoSql database
  2. Strongly consistent read and write
  3. Automatic sharding
  4. Automatic RegionServer failover
  5. Hadoop/HDFS Integration
  6. HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
  7. HBase supports an easy to use Java API for programmatic access.
  8. HBase also supports Thrift and REST for non-Java front-ends.

Hadoop Environment

HIVE

- SQL-like queries and tables on large datasets
- **Features of HIVE:-**
  1. An sql like interface to Hadoop.
  2. Data warehouse infrastructure built on top of Hadoop
  3. Provide data summarization, query and analysis
  4. Query execution via MapReduce
  5. Hive interpreter convert the query to Map reduce format.
  6. Open source project.
  7. Developed by Facebook
  8. Also used by Netflix, Cnet, Digg, eHarmony etc.

Hadoop Environment

PIG

- Data flow language and compiler
- **Features of pig:-**
  1. A scripting platform for processing and analyzing large data sets
  2. Apache Pig allows to write complex MapReduce programs using a simple scripting language.
  3. High level language: Pig Latin
  4. Pig Latin is data flow language.
  5. Pig translate Pig Latin script into MapReduce to execute within Hadoop.
  6. Open source project
  7. Developed by Yahoo

Hadoop Environment

ZOOKEEPER

- Coordination service for distributed applications
- **Features of Zookeeper:-**

  1. Because coordinating distributed systems is a Zoo.
  2. ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Hadoop Environment

FLUME

- Configurable streaming data collection
- Apache Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of streaming data into the Hadoop Distributed File System (HDFS).

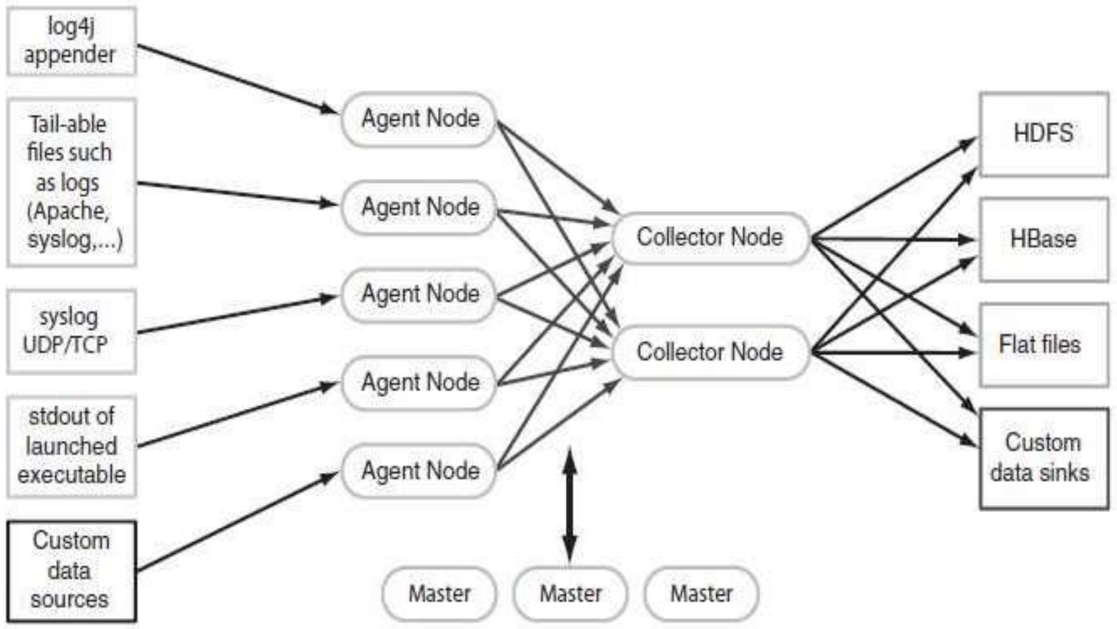
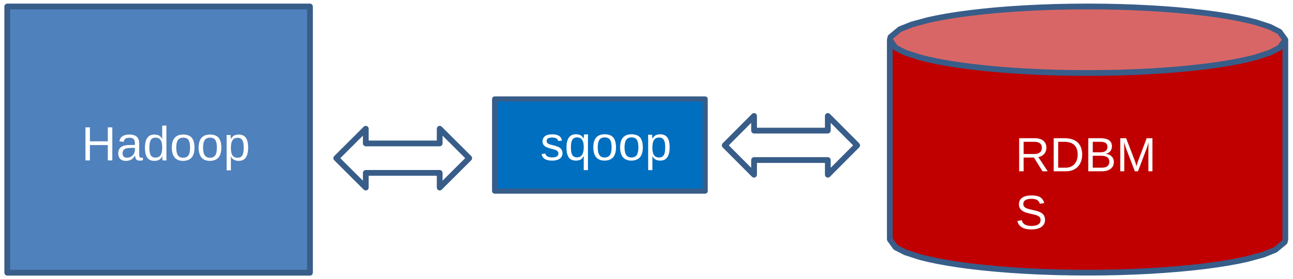


Figure 2.2 Flume architecture for collecting streaming data

Hadoop Environment

SQOOP

- Integration of databases and data warehouses with Hadoop
- **Features of Sqoop:-**
  1. Command-line interface for transforming data between relational database and Hadoop
  2. Support incremental imports
  3. Imports use to populate tables in Hadoop
  4. Exports use to put data from Hadoop into relational database such as SQL server



Hadoop Environment

OOZIE

- To design and schedule workflows
- Oozie is a workflow scheduler where the workflows are expressed as Directed Acyclic Graphs. Oozie runs in a Java servlet container Tomcat and makes use of a database to store all the running workflow instances, their states ad variables along with the workflow definitions to manage Hadoop jobs (MapReduce, Sqoop, Pig and Hive).The workflows in Oozie are executed based on data and time dependencies.

Benefits and challenges

Objective:

- **In this topic** we focus on Advantages of Hadoop Varied Data Sources.
- Hadoop accepts a variety of data. ...
- Cost-effective. Hadoop is an economical solution as it uses a cluster of commodity hardware to store data. ...
- Performance. ...
- Fault-Tolerant. ...
- Highly Available. ...
- Low Network Traffic. ...
- High Throughput. ...
- Open Source.

Recap:

- Revision of Hadoop Architecture.

Benefits and challenges

Hadoop Advantages

- Unlimited data storage

1. Server Scaling Mode

   a) Vertical Scale b)Horizontal Scale

- High speed processing system
- All varities of data processing
  1. Structural
  2. Unstructural
  3. semi-structural

Benefits and challenges

Disadvantage of Hadoop

- If volume is small then speed of hadoop is bad
- Limitation of hadoop data storage

  Well there is obviously a practical limit. But physically HDFS Block IDs are Java longs so they have a max of 2^63 and if your block size is 64 MB then the maximum size is 512 yottabytes.

- Hadoop should be used for only batch processing

  1. Batch process:-background process

     where user can't interactive

- Hadoop is not used for OLTP
  – OLTP process:-interactive with uses

Benefits and challenges

3 Main Challenges of Hadoop:

- **Challenge** #1: The Difficulty in Finding Root Cause of Problems.
- **Challenge** #2: Inefficient Cluster Utilization.
- **Challenge** #3: The business impact of **Hadoop** inefficiencies.

Hadoop file system interfaces

Objective:

- **In this topic** we focus on One of Hadoop’s key features is its implementation of a distributed file system. A distributed file system offers a logical file system that is implemented over a cluster of machines rather than on a single machine. Additional details on HDFS are available on the Apache Hadoop website. Hadoop offers access to HDFS through a command line interface.

Recap:

- Revision of Hadoop File Systems.

Hadoop file system interfaces

- Hadoop has an abstract notion of filesystems, of which HDFS is just one implementation.
- The Java abstract class org.apache.hadoop.fs.FileSystem represents the client
- interface to a filesystem in Hadoop, and there are several concrete implementations.
- The main ones that ship with Hadoop are described
- to pick the correct filesystem instance to communicate with. For example, the filesystem
- shell that we met in the previous section operates with all Hadoop filesystems. To list
- the files in the root directory of the local filesystem, type:
- % hadoop fs -ls file:///
- Although it is possible (and sometimes very convenient) to run MapReduce programs
- that access any of these filesystems, when you are processing large volumes of data you
- should choose a distributed filesystem that has the data locality optimization, notably
- HDFS

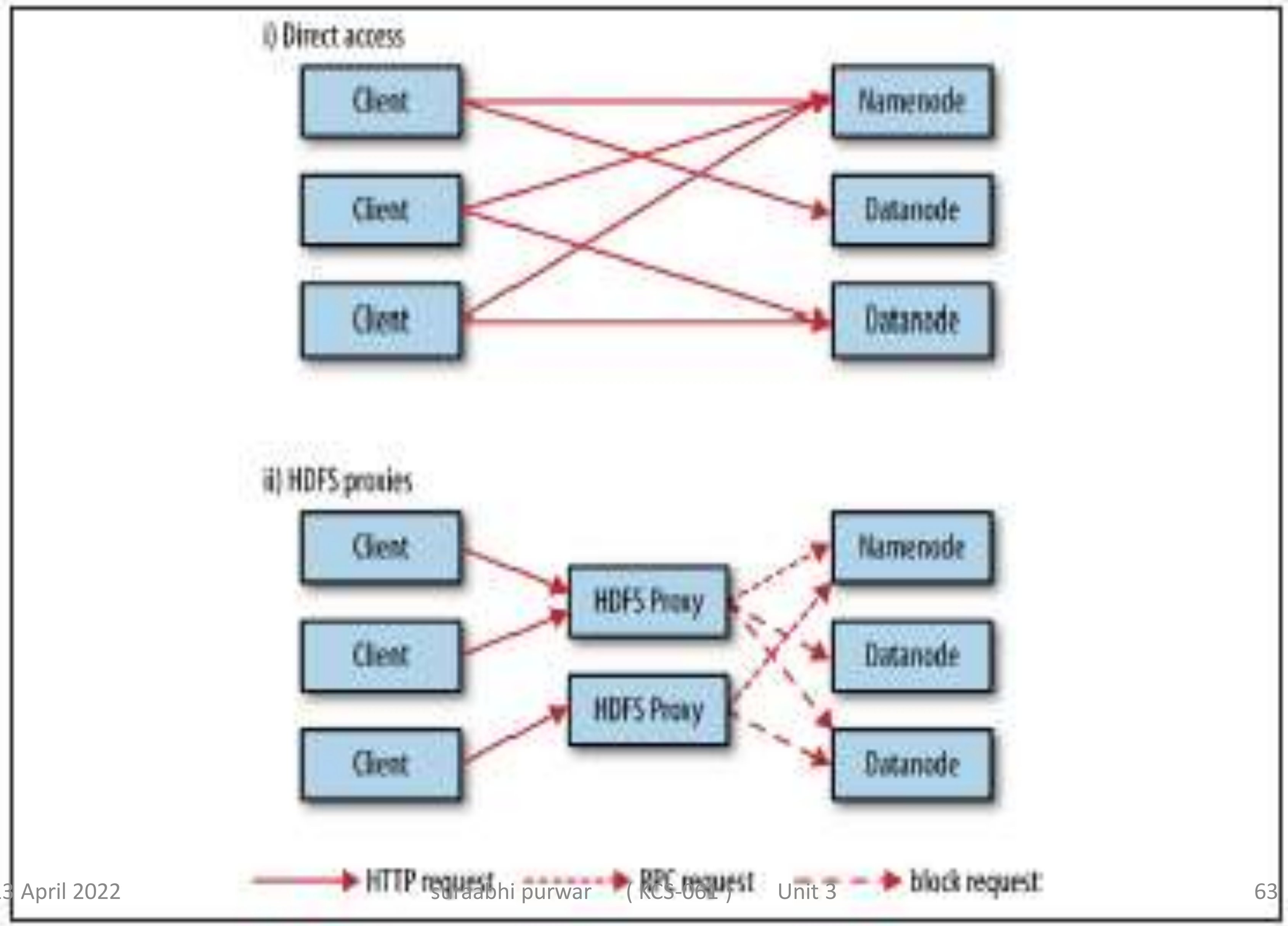
Hadoop file system interfaces

- **Interfaces**
- Hadoop is written in Java, so most Hadoop filesystem interactions are mediated through
- the Java API. The filesystem shell, for example, is a Java application that uses the Java
- FileSystem class to provide filesystem operations. The other filesystem interfaces are
- discussed briefly in this section. These interfaces are most commonly used with HDFS,
- since the other filesystems in Hadoop typically have existing tools to access the underlying
- filesystem (FTP clients for FTP, S3 tools for S3, etc.), but many of them will work
- with any Hadoop filesystem.

Hadoop file system interfaces

- HTTP
- By exposing its filesystem interface as a Java API, Hadoop makes it awkward for non-
- Java applications to access HDFS. The HTTP REST API exposed by the WebHDFS
- protocol makes it easier for other languages to interact with HDFS. Note that the HTTP
- interface is slower than the native Java client, so should be avoided for very large data
- transfers if possible.
- There are two ways of accessing HDFS over HTTP: directly, where the HDFS daemons
- serve HTTP requests to clients; and via a proxy (or proxies), which accesses HDFS on
- the client's behalf using the usual DistributedFileSystem API. The two ways are illustrated
- in Figure. Both use the WebHDFS protocol.

Hadoop file system interfaces



Hadoop file system interfaces

- **C**
- Hadoop provides a C library called *libhdfs* that mirrors the Java FileSystem interface
- (it was written as a C library for accessing HDFS, but despite its name it can be used to access any Hadoop filesystem). It works using the *Java Native Interface* (JNI) to call a Java filesystem client.
- **NFS**
- It is possible to mount HDFS on a local client's filesystem using Hadoop's NFSv3 gateway.
- You can then use Unix utilities (such as ls and cat) to interact with the filesystem,
- upload files, and in general use POSIX libraries to access the filesystem from any programming
- language. Appending to a file works, but random modifications of a file do
- not, since HDFS can only write to the end of a file.
- **FUSE**
- *Filesystem in Userspace* (FUSE) allows filesystems that are implemented in user space
- to be integrated as Unix filesystems. Hadoop's Fuse-DFS contrib module allows HDFS
- (or any Hadoop filesystem) to be mounted as a standard local filesystem. Fuse-DFS is
- implemented in C using *libhdfs* as the interface to HDFS. At the time of writing, the
- Hadoop NFS gateway is the more robust solution to mounting HDFS, so should be
- preferred over Fuse-DFS.

Hadoop file system interfaces

- **The Java Interface**
- In this section, we dig into the Hadoop FileSystem class: the API for interacting with one of Hadoop's filesystems.6 Although we focus mainly on the HDFS implementation,
- DistributedFileSystem, in general you should strive to write your code against the
- FileSystem abstract class, to retain portability across filesystems.
- This is very useful when testing your program, for example, because you can rapidly run tests using data stored on the local filesystem.

Data flow, data ingest with Flume and Scoop

Objective:

- **In this topic** we focus on Big data systems are popular for processing huge amounts of unstructured data from multiple data sources. ... The major difference between Sqoop and Flume is that Sqoop is used for loading data from relational databases into HDFS while Flume is used to capture a stream of moving data.

Recap:

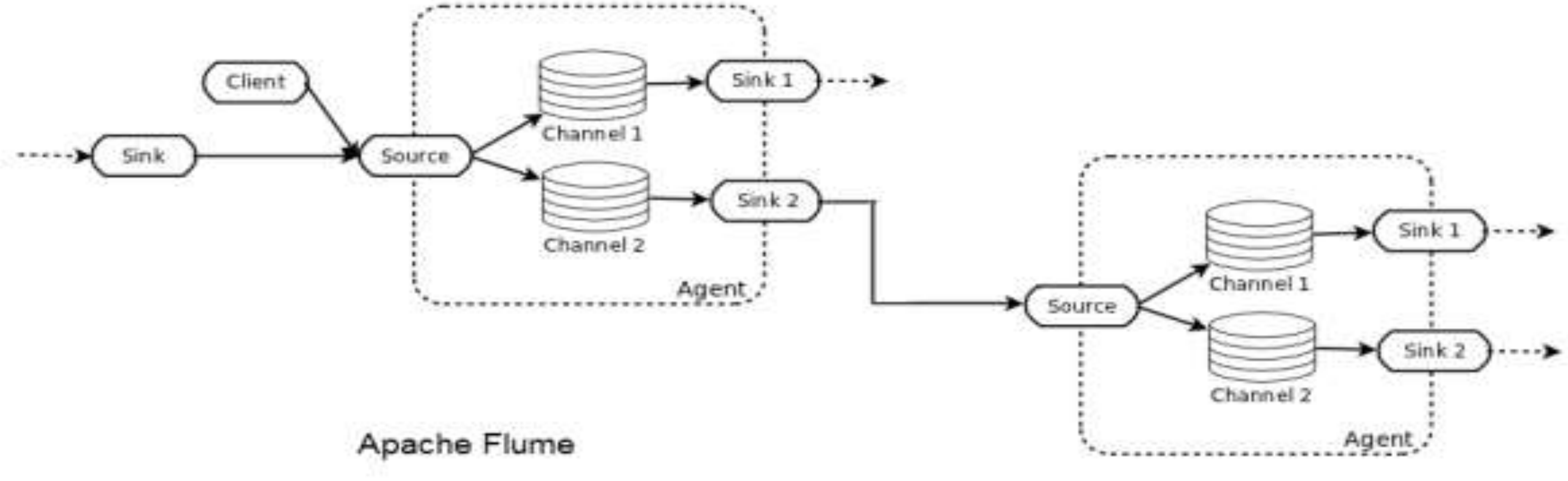
- Revision of Hadoop ecosystems..

Data flow, data ingest with Flume and Scoop

- **Flume** is a **data ingestion** tool that moves **data** from one place to another. In Kafka, the **Flume** is integrated for streaming a high volume of **data** logs from Source to Destination for Storing **data** in HDFS.
- When building big data pipelines, we need to think on how to ingest the volume, variety, and velocity of data showing up at the gates of what would typically be a Hadoop ecosystem. Preliminary considerations such as scalability, reliability, adaptability, cost in terms of development time, etc. will all come into play when deciding on which tools to adopt to meet our requirements. In this article, we'll focus briefly on three Apache ingestion tools: Flume, Kafka, and NiFi. All three products offer great performance, can be scaled horizontally, and provide a plug-in architecture where functionality can be extended through custom components.

Data flow, data ingest with Flume and Scoop

- A Flume deployment consists of one or more agents configured with a topology. The Flume Agent is a JVM process that hosts the basic building blocks of a Flume topology, which are the source, the channel, and the sink. Flume clients send events to the source, which places those events in batches into a temporary buffer called channel, and from there the data flows to a sink connecting to data's final destination. A sink can also be a follow-on source of data for other Flume agents. Agents can be chained and have each multiple sources, channels, and sinks.



Data flow, data ingest with Flume and Scoop

- Flume is a distributed system that can be used to collect, aggregate, and transfer streaming events into Hadoop. It comes with many built-in sources, channels, and sinks, for example, Kafka Channel and Avro sink. Flume is configuration-based and has interceptors to perform simple transformations on in-flight data.

- It is easy to lose data using Flume if you're not careful. For instance, choosing the memory channel for high throughput has the downside that data will be lost when the agent node goes down. A file channel will provide durability at the price of increased latency. Even then, since data is not replicated to other nodes, the file channel is only as reliable as the underlying disks. Flume does offer scalability through multi-hop/fan-in fan-out flows. For high availability (HA), agents can be scaled horizontally.

Data flow, data ingest with Flume and Scoop

- **SQOOP** is an open source tool that allows you to ingest data from many different types of databases into hdfs. It also has the ability to export data from hdfs back into an external database.
- **Importing a single table**
- I will be using a MySQL database as my external source to import data into hdfs.
- To import a single table, as a minimum, SQOOP requires two parameters: a connection string and a table. To go with the connection string, you may need to specify a username and password.
- sqoop-import --connect jdbc:mysql://localhost:3306/test --username user --password 1234 --table test_table
- This will import all rows and columns in this table and import it into the current user base directory within HDFS as delimited text files, the number of mappers used (default 4) determines the number of output data files. Specifying the target directory
- To specify a target directory for the import, SQOOP has two options. –target-dir or –warehouse-dir. The difference is that –target-dir is a full directory path and the data files will be created directly inside the specified folder. –warehouse-dir is used to specify a base directory within hdfs where SQOOP will create a sub folder inside with the name of the source table, and import the data files into that folder.
- sqoop-import --connect jdbc:mysql://localhost:3306/test --username user --password 1234 --table test_table --target-dir /user/abc/test
- or
- sqoop-import --connect jdbc:mysql://localhost:3306/test --username user --password 1234 --table test_table --warehouse-dir /user/abc
- Note: The target directory MUST NOT already exist, otherwise SQOOP will fail.

Hadoop archives

Objective:

- **In this topic** we focus on Hadoop archive is a facility which packs up small files into one compact HDFSblock to avoid memory wastage of name node.name node stores the metadata information of the the HDFS data.SO,say 1GB file is broken in 1000 pieces then namenode will have to store metadata about all those 1000 small files.

Recap:

- Revision of Hadoop Database systems.

Hadoop archives

- Hadoop archive is a facility which packs up small files into one compact HDFSblock to avoid memory wastage of name node.name node stores the metadata information of the the HDFS data.SO,say 1GB file is broken in 1000 pieces then namenode will have to store metadata about all those 1000 small files.In that manner,namenode memory willbe wasted it storing and managing a lot of data.
- HAR is created from a collection of files and the archiving tool will run a MapReduce job.these Maps reduce jobs to process the input files in parallel to create an archive file.

HAR command

```
hadoop archive -archiveName myhar.har /input/location /output/location
```

Hadoop is created to deal with large files data .So small files are problematic and to be handled efficiently.
As large input file is splitted into number of small input files and stored across all the data nodes, all these huge number of records are to be stored in name node which makes name node inefficient. To handle this problem, Hadoop Archieve has been created which packs the HDFS files into archives and we can directly use these files an as input to the MR jobs. It always comes with *.har extension.

Hadoop archives

- **HAR Syntax:**
- hadoop archive -archiveName NAME -p <parent path> <src>* <dest>
- Example:
- hadoop archive -archiveName foo.har -p /user/hadoop dir1 dir2 /user/zoo
- If you have a hadoop archive stored in HDFS in /user/zoo/foo.har then for using this archive for MapReduce input, all you need to specify the input directory as har:///user/zoo/foo.har.
- If we list the archive file:
- $hadoop fs -ls /data/myArch.har
- /data/myArch..har/_index
- /data/myArch..har/_masterindex
- /data/myArch..har/part-0
- part files are the original files concatenated together with big files and index files are to look up for the small files in the big part file.
- **Limitations of HAR Files:**
- 1) Creation of HAR files will create a copy of the original files. So, we need as much disk space as size of original files which we are archiving.We can delete the original files after creation of archive to release some disk space.
- 2) Once an archive is created, to add or remove files from/to archive we need to re-create the archive.
- 3) HAR file will require lots of map tasks which are inefficient.

File sizes, block sizes and block abstraction in HDFS

Objective:

- **In this topic** we focus on 128 MB
- A typical block size used by HDFS is 128 MB. Thus, an HDFS file is chopped up into 128 MB chunks, and if possible, each chunk will reside on a different DataNode.

Recap:

- Revision of Hadoop Architecture.

File sizes, block sizes and block abstraction in HDFS

Blocks

- A disk has a block size, which is the minimum amount of data that it can read or write.
- Filesystems for a single disk build on this by dealing with data in blocks, which are an
- integral multiple of the disk block size. Filesystem blocks are typically a few kilobytes
- in size, whereas disk blocks are normally 512 bytes. This is generally transparent to the
- filesystem user who is simply reading or writing a file of whatever length. However,
- there are tools to perform filesystem maintenance, such as *df* and *fsck*, that operate on
- the filesystem block level.
- HDFS, too, has the concept of a block, but it is a much larger unit—128 MB by default.
- Like in a filesystem for a single disk, files in HDFS are broken into block-sized chunks,
- which are stored as independent units. Unlike a filesystem for a single disk, a file in
- HDFS that is smaller than a single block does not occupy a full block’s worth of underlying
- storage. (For example, a 1 MB file stored with a block size of 128 MB uses 1 MB
- of disk space, not 128 MB.) When unqualified, the term “block” in this book refers to a
- block in HDFS.

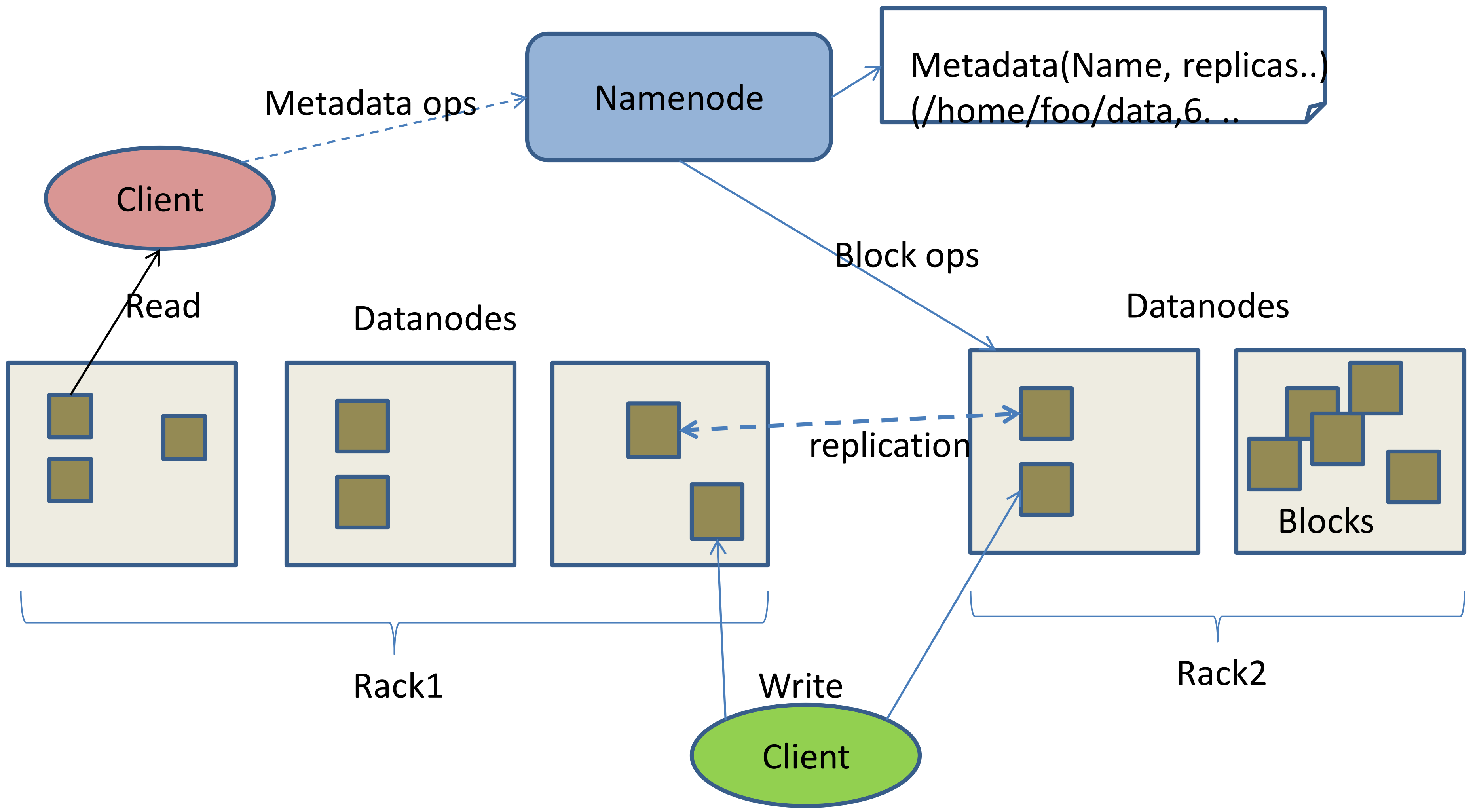
File sizes, block sizes and block abstraction in HDFS

Namenode and Datanodes

- Master/slave architecture
- HDFS cluster consists of a single **Namenode**, a master server that manages the file system namespace and regulates access to files by clients.
- There are a number of **DataNodes** usually one per node in a cluster.
- The DataNodes manage storage attached to the nodes that they run on.
- HDFS exposes a file system namespace and allows user data to be stored in files.
- A file is split into one or more blocks and set of blocks are stored in DataNodes.
- DataNodes: serves read, write requests, performs block creation, deletion, and replication upon instruction from Namenode.

File sizes, block sizes and block abstraction in HDFS

HDFS Architecture



File sizes, block sizes and block abstraction in HDFS

File system Namespace

- Hierarchical file system with directories and files
- Create, remove, move, rename etc.
- Namenode maintains the file system
- Any meta information changes to the file system recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

File sizes, block sizes and block abstraction in HDFS

Safemode Startup

- On startup Namenode enters Safemode.
- Replication of data blocks do not occur in Safemode.
- Each DataNode checks in with Heartbeat and BlockReport.
- Namenode verifies that each block has acceptable number of replicas
- After a configurable percentage of safely replicated blocks check in with the Namenode, Namenode exits Safemode.
- It then makes the list of blocks that need to be replicated.
- Namenode then proceeds to replicate these blocks to other Datanodes.

File sizes, block sizes and block abstraction in HDFS

Filesystem Metadata

- The HDFS namespace is stored by Namenode.
- Namenode uses a transaction log called the EditLog to record every change that occurs to the filesystem meta data.
  - For example, creating a new file.
  - Change replication factor of a file
  - EditLog is stored in the Namenode's local filesystem
- Entire filesystem namespace including mapping of blocks to files and file system properties is stored in a file FsImage. Stored in Namenode's local filesystem.

File sizes, block sizes and block abstraction in HDFS

Namenode

- Keeps image of entire file system namespace and file Blockmap in memory.
- 4GB of local RAM is sufficient to support the above data structures that represent the huge number of files and directories.
- When the Namenode starts up it gets the FsImage and Editlog from its local file system, update FsImage with EditLog information and then stores a copy of the FsImage on the filesytstem as a checkpoint.
- Periodic checkpointing is done. So that the system can recover back to the last checkpointed state in case of a crash.

File sizes, block sizes and block abstraction in HDFS

Datanode

- A Datanode stores data in files in its local file system.
- Datanode has no knowledge about HDFS filesystem
- It stores each block of HDFS data in a separate file.
- Datanode does not create all files in the same directory.
- It uses heuristics to determine optimal number of files per directory and creates directories appropriately:
    - Research issue?
- When the filesystem starts up it generates a list of all HDFS blocks and send this report to Namenode: Blockreport.

File sizes, block sizes and block abstraction in HDFS

The Communication Protocol

- All HDFS communication protocols are layered on top of the TCP/IP protocol
- A client establishes a connection to a configurable TCP port on the Namenode machine. It talks ClientProtocol with the Namenode.
- The Datanodes talk to the Namenode using Datanode protocol.
- RPC abstraction wraps both ClientProtocol and Datanode protocol.
- Namenode is simply a server and never initiates a request; it only responds to RPC requests issued by DataNodes or clients.

Data replication

Objective:

- **In this topic** we focus on Recap:Data Replication. HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance.

Recap

- Revision of Distributed file systems.

Data replication

Data Replication

- HDFS is designed to store very large files across machines in a large cluster.
- Each file is a sequence of blocks.
- All blocks in the file except the last are of the same size.
- Blocks are replicated for fault tolerance.
- Block size and replicas are configurable per file.
- The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.
- BlockReport contains all the blocks on a Datanode.

Data replication

Replica Placement

- The placement of the replicas is critical to HDFS reliability and performance.
- Optimizing replica placement distinguishes HDFS from other distributed file systems.
- Rack-aware replica placement:
  - Goal: improve reliability, availability and network bandwidth utilization
  - Research topic
- Many racks, communication between racks are through switches.
- Network bandwidth between machines on the same rack is greater than those in different racks.
- Namenode determines the rack id for each DataNode.
- Replicas are typically placed on unique racks
  - Simple but non-optimal
  - Writes are expensive
  - Replication factor is 3
  - Another research topic?
- Replicas are placed: one on a node in a local rack, one on a different node in the local rack and one on a node in a different rack.
- 1/3 of the replica on a node, 2/3 on a rack and 1/3 distributed evenly across remaining racks.

Data replication

Replica Selection

- Replica selection for READ operation: HDFS tries to minimize the bandwidth consumption and latency.
- If there is a replica on the Reader node then that is preferred.
- HDFS cluster may span multiple data centers: replica in the local data center is preferred over the remote one.

How does HDFS store, read, and write files

Objective:

- **In this topic** we focus on Hadoop HDFS Data Write Operation. To write a file in HDFS, a client needs to interact with master i.e. namenode (master). Now namenode provides the address of the datanodes (slaves) on which client will start writing the data. Client directly writes data on the datanodes, now datanode will create data write pipeline.

Recap:

- Revision of Hadoop Architecture.

How does HDFS store, read, and write files

HDFS Definition-Design of HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.
HDFS is a distributed, scalable, and portable filesystem written in Java for the Hadoop framework.
It has many similarities with existing distributed file systems.
Hadoop Distributed File System (HDFS™) is the primary storage system used by Hadoop applications.
HDFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable, extremely rapid computations.
HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.
HDFS provides high throughput access to application data and is suitable for applications that have large data sets

- HDFS consists of following components (daemons)
  - Name Node
  - Data Node
  - Secondary Name Node

How does HDFS store, read, and write files

HDFS Components

Namenode:

NameNode, a master server, manages the file system namespace and regulates access to files by clients.

Maintains and Manages the blocks which are present on the datanode.

- **Meta-data in Memory**
  – The entire metadata is in main memory
- **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

How does HDFS store, read, and write files

HDFS Components

Data Node: Slaves which are deployed on each mechine and provide the actual storage.

DataNodes, one per node in the cluster, manages storage attached to the nodes that they run on

- **A Block Server**
  - Stores data in the local file system (e.g. ext3)
  - Stores meta-data of a block (e.g. CRC)
  - Serves data and meta-data 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

How does HDFS store, read, and write files

Understanding the File system

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 replica**

Data Correctness

- **Use Checksums to validate data**
  - Use CRC32
- **File Creation**
  - Client computes checksum per 512 byte
  - DataNode stores the checksum
- **File access**
  - Client retrieves the data and checksum from DataNode
  - If Validation fails, Client tries other replicas

How does HDFS store, read, and write files

Understanding the File system

- **Data pipelining**
  - 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 DataNode in the Pipeline
  - When all replicas are written, the Client moves on to write the next block in file

Rebalancer

- **Goal: % of disk occupied 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

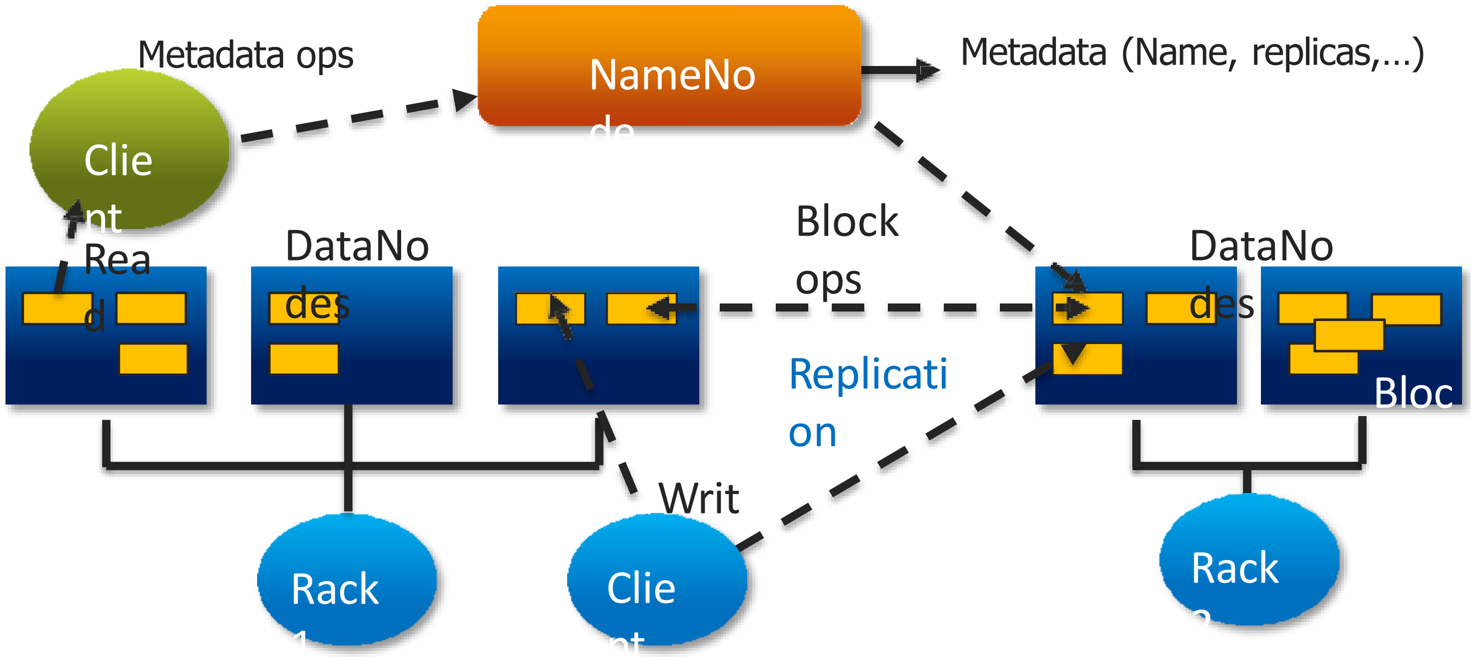
How does HDFS store, read, and write files

Secondary NameNode

- ✓ Secondary NameNode:
  - ✓ Not a hot standby for the NameNode
  - ✓ Connects to NameNode every hour*
  - ✓ Housekeeping, backup of NemeNode metadata
  - ✓ Saved metadata can build a failed NameNode

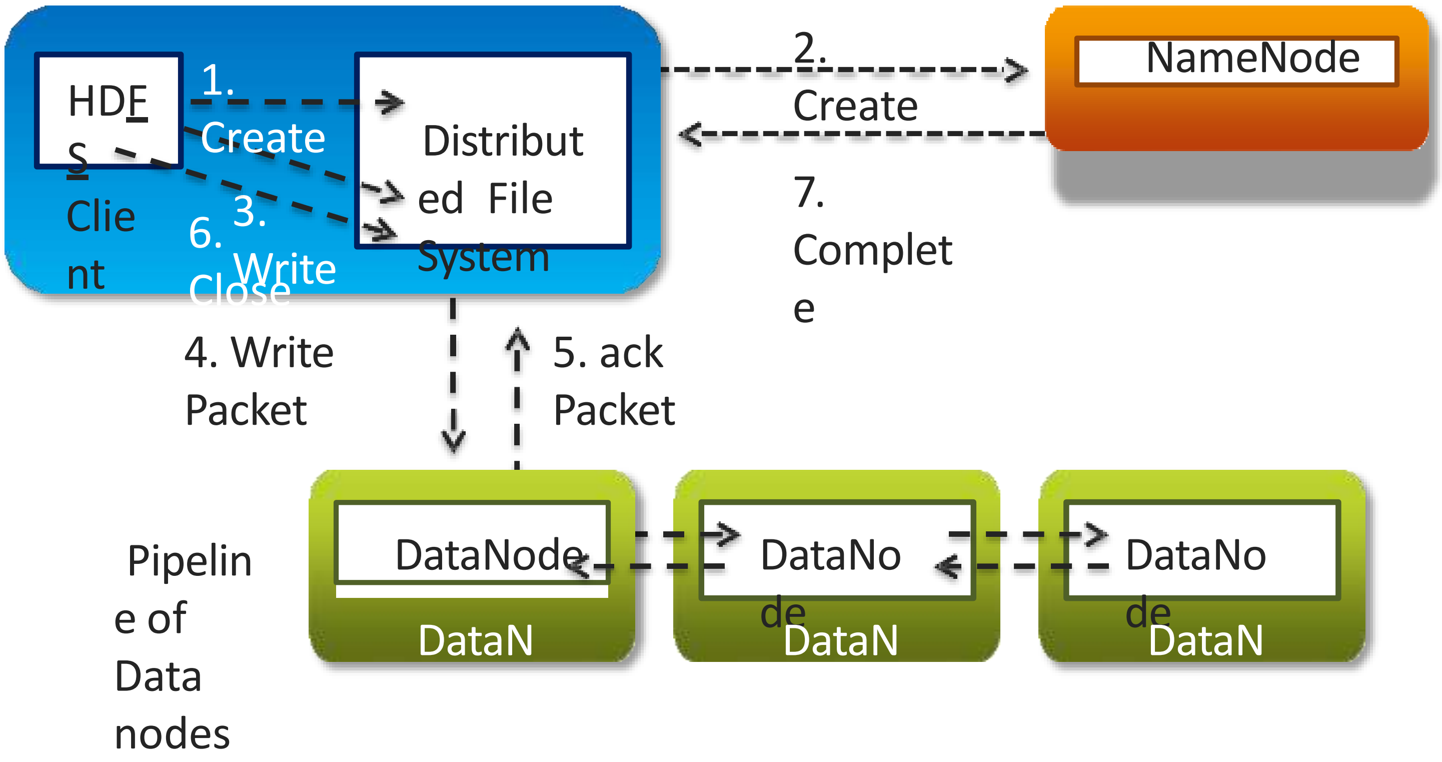
How does HDFS store, read, and write files

HDFS Architecture



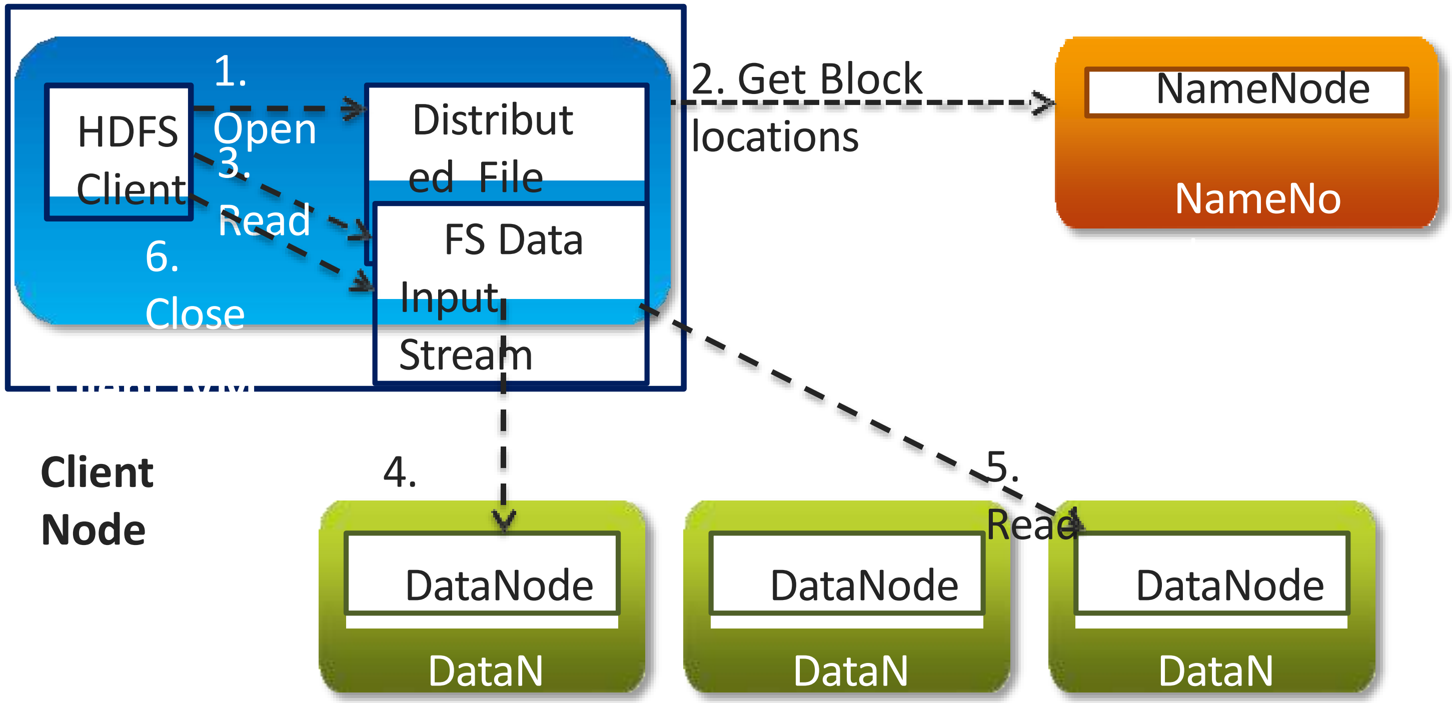
How does HDFS store, read, and write files

Anatomy of A File Write



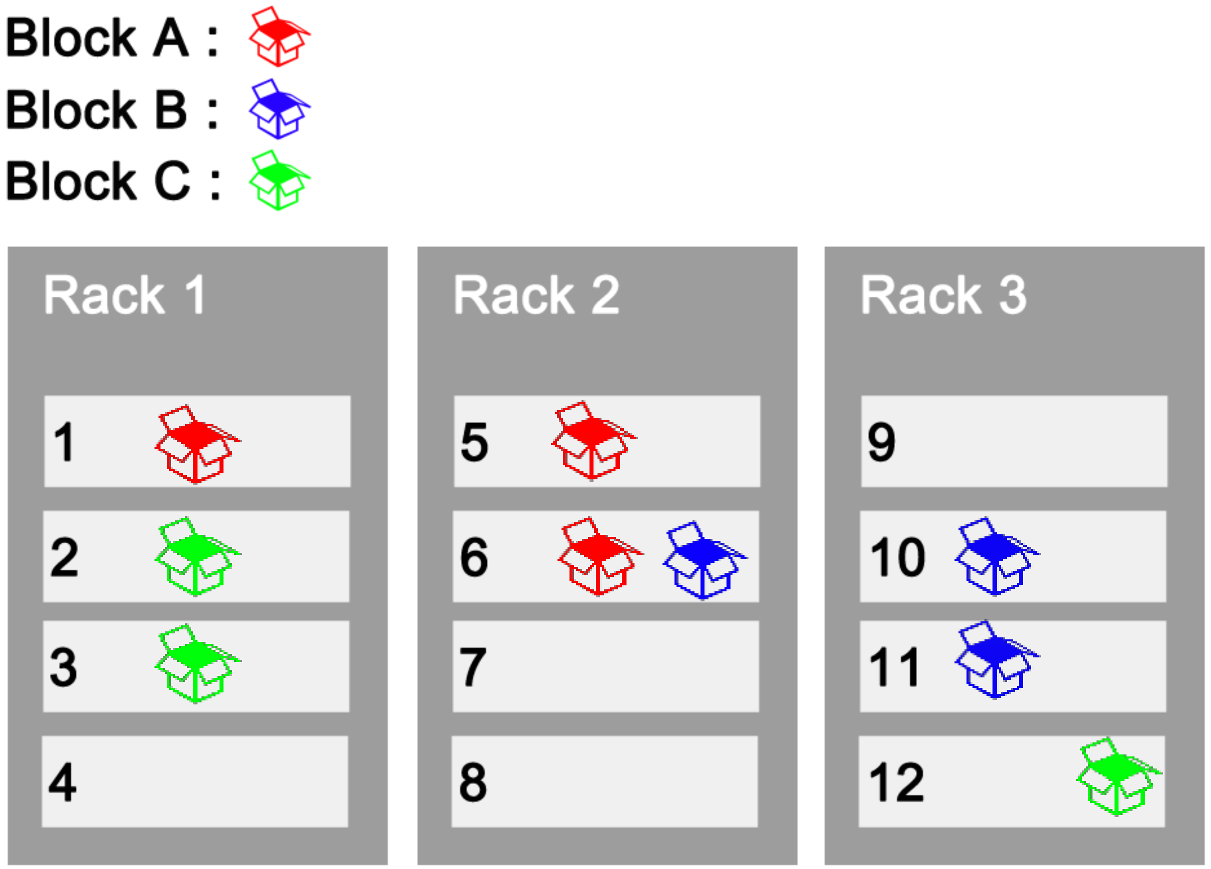
How does HDFS store, read, and write files

Anatomy of A File Read



How does HDFS store, read, and write files

Replication and Rack Awareness



Setting up a Hadoop cluster, Cluster specification, cluster setup and installation

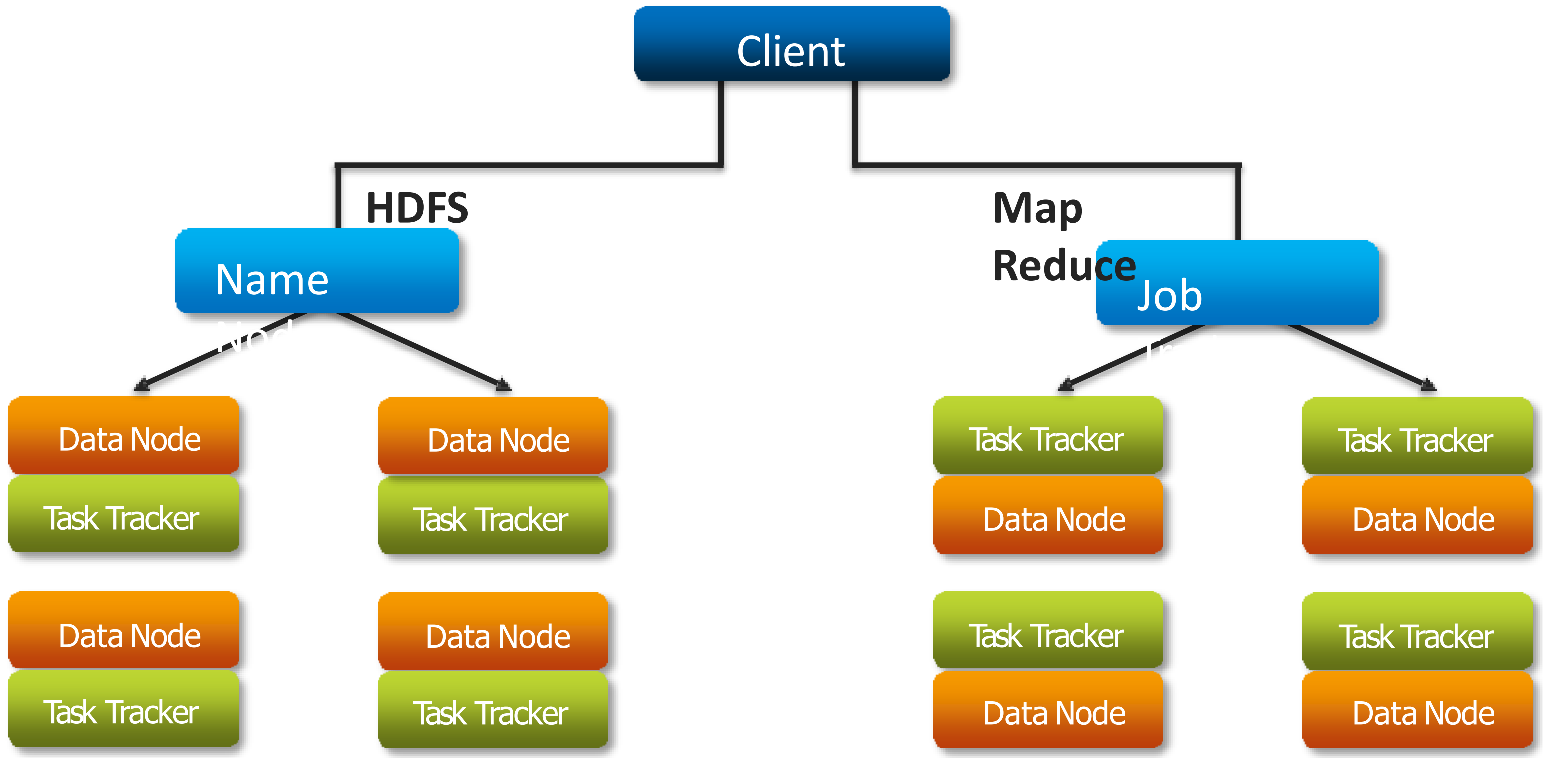
Objective:

- **In this topic** we focus on Hadoop cluster you will need to configure the environment in which the Hadoop daemons execute as well as the configuration parameters for the Hadoop daemons. HDFS daemons are NameNode, SecondaryNameNode, and DataNode. YARN daemons are ResourceManager, NodeManager, and WebAppProxy.

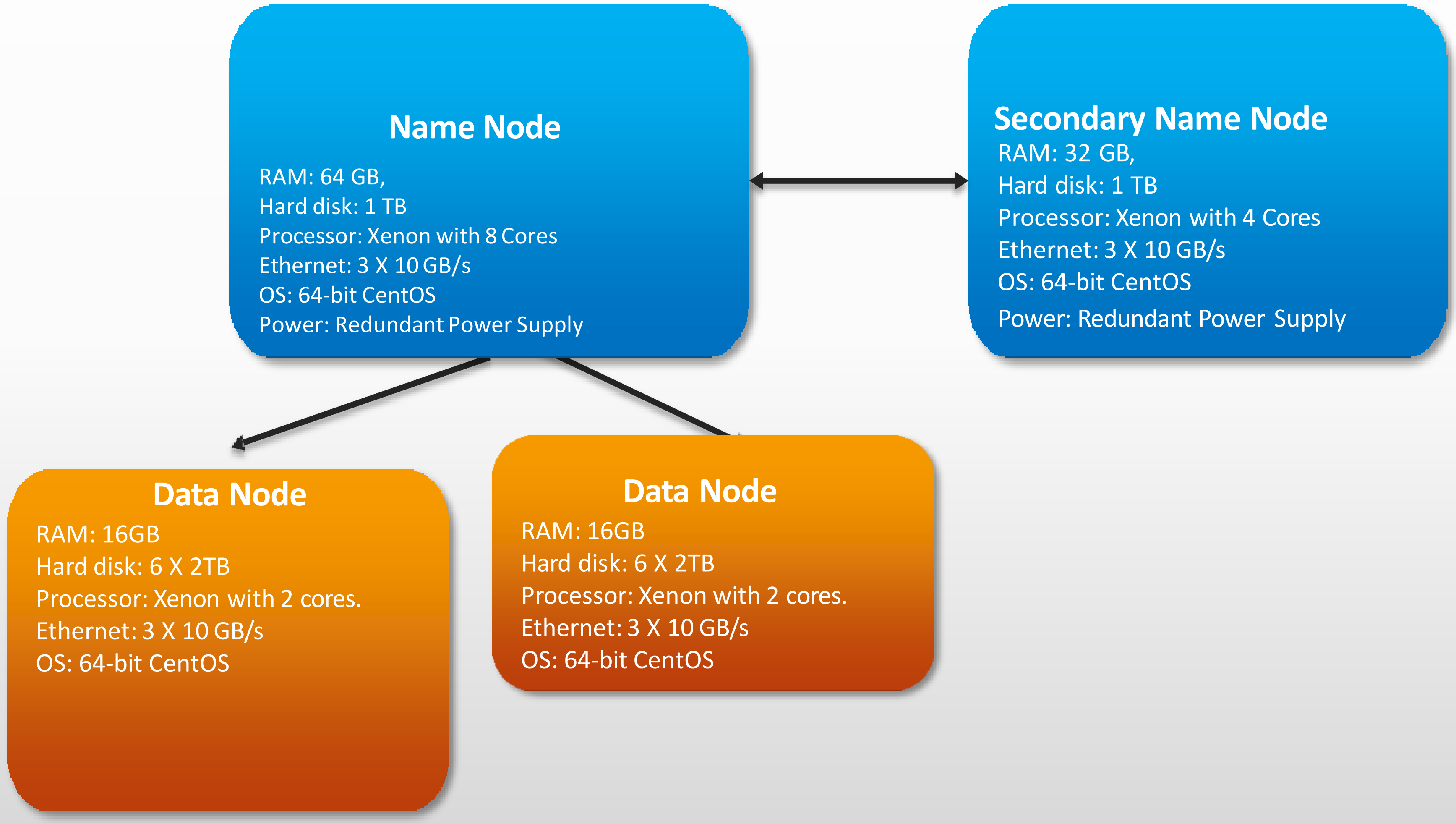
Recap:

- Revision of Hadoop Architecture.

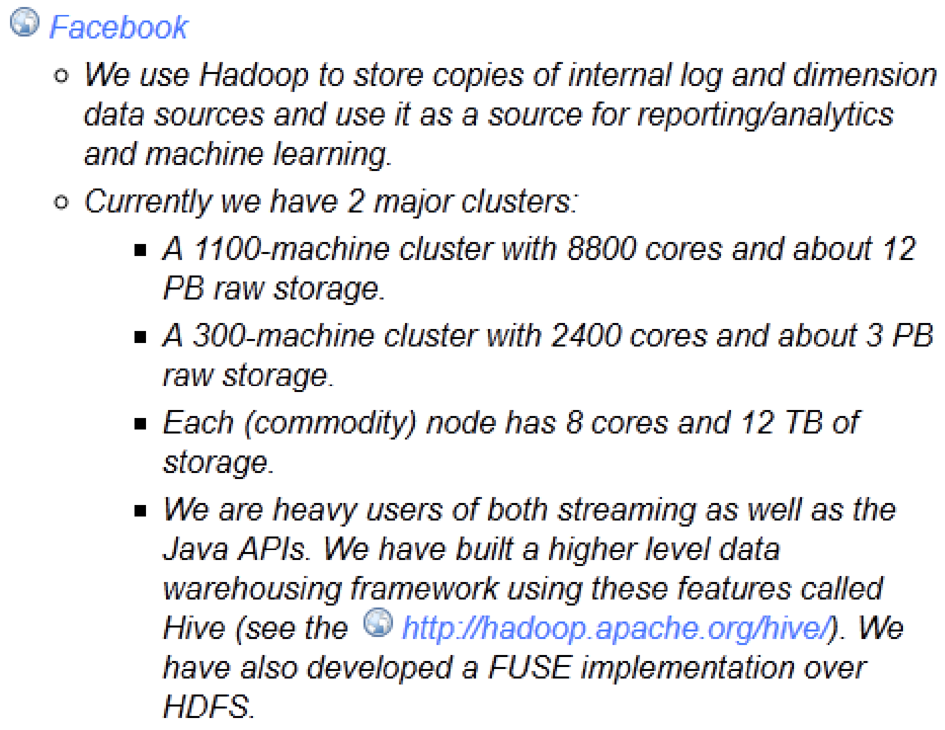
Hadoop Cluster Architecture (Contd.)



Hadoop Cluster: A Typical Use Case



Hadoop Cluster: Facebook



Facebook

- *We use Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning.*
- *Currently we have 2 major clusters:*
  - *A 1100-machine cluster with 8800 cores and about 12 PB raw storage.*
  - *A 300-machine cluster with 2400 cores and about 3 PB raw storage.*
  - *Each (commodity) node has 8 cores and 12 TB of storage.*
  - *We are heavy users of both streaming as well as the Java APIs. We have built a higher level data warehousing framework using these features called Hive (see the http://hadoop.apache.org/hive/). We have also developed a FUSE implementation over HDFS.*

http://wiki.apache.org/hadoop/PoweredBy

Setting up a Hadoop cluster, Cluster specification, cluster setup and installation

Hadoop Cluster Modes

Hadoop can run in any of the following three modes:

Standalone (or Local) Mode

- ✓ No daemons, everything runs in a single JVM.
- ✓ Suitable for running MapReduce programs during development.
- ✓ Has no DFS.

Pseudo-Distributed Mode

- ✓ Hadoop daemons run on the local machine.

Fully-Distributed Mode

- ✓ Hadoop daemons run on a cluster of machines.

Hadoop configuration

Objective:

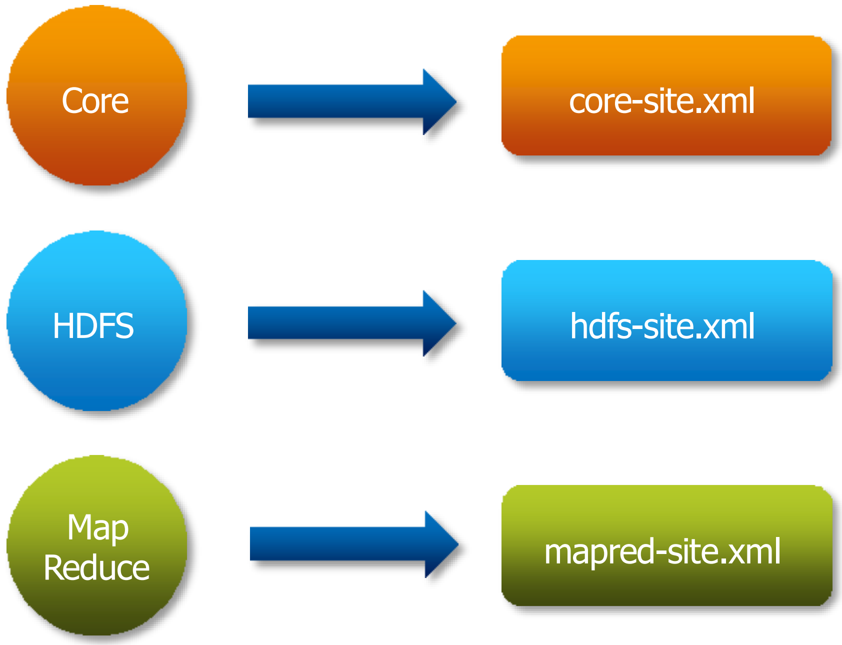
- **In this topic** we focus on Hadoop configuration is driven by two types of important configuration files:
- Read-only default configuration - src/core/core-default. xml, src/hdfs/hdfs-default. xml and src/mapred/mapred-default. xml.
- Site-specific configuration - conf/core-site. xml, conf/hdfs-site. xml and conf/mapred-site. xml.

Recap:

- Revision of Hadoop Filesystem.

Hadoop configuration

Hadoop 1.x: Core Configuration Files



Hadoop configuration

Hadoop Configuration Files

Configuration Filenames	Description of Log Files
hadoop-env.sh	Environment variables that are used in the scripts to run Hadoop.
core-site.xml	Configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce.
hdfs-site.xml	Configuration settings for HDFS daemons, the namenode, the secondary namenode and the data nodes.
mapred-site.xml	Configuration settings for MapReduce daemons : the job-tracker and the task-trackers.
masters	A list of machines (one per line) that each run a secondary namenode.
slaves	A list of machines (one per line) that each run a datanode and a task-tracker.

Hadoop configuration

core-site.xml and hdfs-site.xml

hdfs-site.xml	core-site.xml
<?xml version - "1.0"?>	<?xml version ="1.0"?>
<!--hdfs-site.xml-->	<!--core-site.xml-->
<configuration>	<configuration>
<property>	<property>
<name>**dfs.replication**</name>	<name>**fs.default.name**</name>
<value>**1**</value>	<value>**hdfs://localhost:8020/**</value>
</property>	</property>
</configuration>	</configuration>

Hadoop configuration

Defining HDFS Details In hdfs-site.xml

Property	Value	Description
dfs.data.dir	<value> /disk1/hdfs/data, /disk2/hdfs/data </value>	A list of directories where the datanode stores blocks. Each block is stored in only one of these directories. ${hadoop.tmp.dir}/dfs/data
fs.checkpoint.dir	<value> /disk1/hdfs/namesecondary, /disk2/hdfs/namesecondary </value>	A list of directories where the secondary namenode stores checkpoints. It stores a copy of the checkpoint in each directory in the list ${hadoop.tmp.dir}/dfs/namesecondary

Hadoop configuration

mapred-site.xml

mapred-site.xml
<?xml version="1.0"?>
<configuration>
<property>
<name>**mapred.job.tracker**</name>
<value>**localhost:8021**</value>
<property>
</configuration>

Hadoop configuration

Defining mapred-site.xml

Property	Value	Description
mapred.job.tracker	<value> localhost:8021 </value>	The hostname and the port that the jobtracker RPC server runs on. If set to the default value of local, then the jobtracker runs in-process on demand when you run a MapReduce job.
mapred.local.dir	${hadoop.tmp.dir}/mapred/local	A list of directories where MapReduce stores intermediate data for jobs. The data is cleared out when the job ends.
mapred.system.dir	${hadoop.tmp.dir}/mapred/system	The directory relative to fs.default.name where shared files are stored, during a job run.
mapred.tasktracker.map.tasks.maximum	2	The number of map tasks that may be run on a tasktracker at any one time
mapred.tasktracker.reduce.tasks.maximum	2	The number of reduce tasks tat may be run on a tasktracker at any one time.

Hadoop configuration

All Properties

1. http://hadoop.apache.org/docs/r1.1.2/core-default.html
2. http://hadoop.apache.org/docs/r1.1.2/mapred-default.html
3. http://hadoop.apache.org/docs/r1.1.2/hdfs-default.html

Java interfaces to HDFS, command line interface

Objective:

- **In this topic** we focus on The HDFS can be manipulated through a Java API or through a command line interface. All commands for manipulating HDFS through Hadoop's command line interface begin with "hadoop", a space, and "fs". This is the file system shell. This is followed by the command name as an argument to "hadoop fs".

Recap:

- Revision of interfaces.

Java interfaces to HDFS, command line interface

Slaves and Masters

Two files are used by the startup and shutdown commands:

Slaves

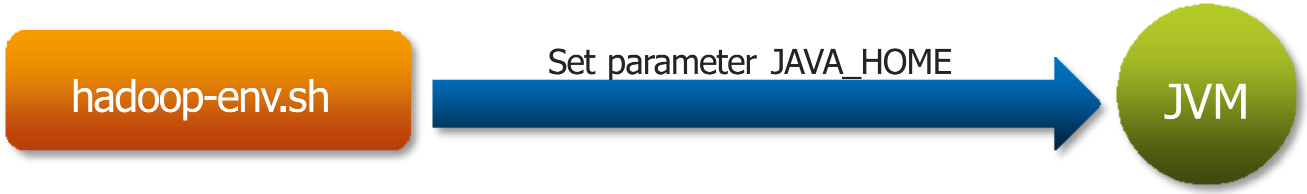
- ✓ Contains a list of hosts, one per line, that are to host **DataNode** and **TaskTracker** servers.

Masters

- ✓ Contains a list of hosts, one per line, that are to host **Secondary NameNode** servers.

Java interfaces to HDFS, command line interface

Per-Process Runtime Environment



- ✓ This file also offers a way to provide custom parameters for each of the servers.
- ✓ Hadoop-env.sh is sourced by all of the Hadoop Core scripts provided in the conf/ directory of the installation.
- ✓ **Examples of environment variables that you can specify:**

```
export HADOOP_DATANODE_HEAPSIZE="128"

export HADOOP_TASKTRACKER_HEAPSIZE="512"
```

Java interfaces to HDFS, command line interface

Web UI URLs

- ✓ NameNode status: http://localhost:50070/dfshealth.jsp
- ✓ JobTracker status: http://localhost:50030/jobtracker.jsp
- ✓ TaskTracker status: http://localhost:50060/tasktracker.jsp
- ✓ DataBlock Scanner Report: http://localhost:50075/blockScannerReport

Java interfaces to HDFS, command line interface

Open a terminal window to the current working directory.
# /home/notroot
- -
# 1. Print the Hadoop version
hadoop version
- -
# 2. List the contents of the root directory in HDFS
hadoop fs -ls /
- -
# 3. Report the amount of space used and available on currently mounted filesystem
hadoop fs -df hdfs:/
- -
# 4. Count the number of directories,files and bytes under the paths that match the specified file pattern
hadoop fs -count hdfs:/

Java interfaces to HDFS, command line interface

5. Run a DFS filesystem checking utility
hadoop fsck – /

# 6. Run a cluster balancing utility
hadoop balancer

# 7. Create a new directory named “hadoop” below the /user/training directory in HDFS.
hadoop fs -mkdir /user/training/hadoop

# 8. Add a sample text file from the local directory named “data” to the new directory you created in HDFS
hadoop fs -put data/sample.txt /user/training/hadoop

Java interfaces to HDFS, command line interface

9. List the contents of this new directory in HDFS.

```
hadoop fs -ls /user/training/hadoop
```

- -

# 10. Add the entire local directory called "retail" to the /user/training directory in HDFS.

```
hadoop fs -put data/retail /user/training/hadoop
```

- -

# 11. Since /user/training is your home directory in HDFS, any command that does not have an absolute path is interpreted as relative to that directory.
# The next command will therefore list your home directory, and should show the items you've just added there.

```
hadoop fs -ls
```

- -

# 12. See how much space this directory occupies in HDFS.

```
hadoop fs -du -s -h hadoop/retail
```

Java interfaces to HDFS, command line interface

13. Delete a file ‘customers’ from the “retail” directory.
hadoop fs -rm hadoop/retail/customers

# 14. Ensure this file is no longer in HDFS.
hadoop fs -ls hadoop/retail/customers

# 15. Delete all files from the “retail” directory using a wildcard.
hadoop fs -rm hadoop/retail/*

# 16. To empty the trash
hadoop fs –expunge

# 17. Finally, remove the entire retail directory and all of its contents in HDFS.
hadoop fs -rm -r hadoop/retail

Java interfaces to HDFS, command line interface

# 18. Add the purchases.txt file from the local directory named “/home/training/” to the hadoop directory you created in HDFS

```
hadoop fs -copyFromLocal /home/training/purchases.txt hadoop/
```

- -

# 19. To view the contents of your text file purchases.txt which is present in your hadoop directory.

```
hadoop fs -cat hadoop/purchases.txt
```

- -

# 20. Move a directory from one location to other

```
hadoop fs -mv hadoop apache_hadoop
```

- -

# 21. Add the purchases.txt file from “hadoop” directory which is present in HDFS directory to the directory “data” which is present in your local directory

```
hadoop fs -copyToLocal hadoop/purchases.txt /home/training/data
```

Java interfaces to HDFS, command line interface

```
# 26. Default names of owner and group are training,training
# Use '-chown' to change owner name and group name simultaneously
hadoop fs -ls hadoop/purchases.txt
sudo -u hdfs hadoop fs -chown root:root hadoop/purchases.txt
-------------------------------------------
# 27. Default name of group is training
# Use '-chgrp' command to change group name
hadoop fs -ls hadoop/purchases.txt
sudo -u hdfs hadoop fs -chgrp training hadoop/purchases.txt
-------------------------------------------
# 28. Default replication factor to a file is 3.
# Use '-setrep' command to change replication factor of a file
hadoop fs -setrep -w 2 apache_hadoop/sample.txt
```

Java interfaces to HDFS, command line interface

29. Copy a directory from one node in the cluster to another
# Use ‘-distcp’ command to copy,
# Use '-overwrite' option to overwrite in an existing files
# Use '-update' command to synchronize both directories
hadoop fs -distcp hdfs://namenodeA/apache_hadoop hdfs://namenodeB/hadoop

- -

# 30. Command to make the name node leave safe mode
hadoop fs -expunge
sudo -u hdfs hdfs dfsadmin -safemode leave

- -

# 31. List all the hadoop file system shell commands
hadoop fs

- -

# 33. Last but not least, always ask for help!
hadoop fs -help

Hadoop I/O: compression, serialization

Objective:

- **In this topic** we focus on Hadoop I/O Hadoop comes with a set of primitives for data I/O. Some of these are ... that are more general than Hadoop, such as data integrity and compression.

Recap:

- Revision of Hadoop Architecture.

Hadoop I/O: compression, serialization

Hadoop comes with a set of primitives for data I/O. Some of these are techniques that are more general than Hadoop, such as data integrity and compression, but deserve special consideration when dealing with multiterabyte datasets. Others are Hadoop tools or APIs that form the building blocks for developing distributed systems, such as serialization frameworks and on-disk data structures.

Data Integrity

Users of Hadoop rightly expect that no data will be lost or corrupted during storage or processing. However, since every I/O operation on the disk or network carries with it a small chance of introducing errors into the data that it is reading or writing, when the volumes of data flowing through the system are as large as the ones Hadoop is capable of handling, the chance of data corruption occurring is high.

The usual way of detecting corrupted data is by computing a *checksum* for the data when it first enters the system, and again whenever it is transmitted across a channel that is unreliable and hence capable of corrupting the data. The data is deemed to be corrupt if the newly generated checksum doesn't exactly match the original. This technique doesn't offer any way to fix the data—merely error detection. (And this is a reason for not using low-end hardware; in particular, be sure to use ECC memory.) Note that it is possible that it's the checksum that is corrupt, not the data, but this is very unlikely, since the checksum is much smaller than the data.

A commonly used error-detecting code is CRC-32 (cyclic redundancy check), which computes a 32-bit integer checksum for input of any size.

NIET Greater Noida
GET FUTURE READY

Hadoop I/O: compression, serialization

HDFS transparently checksums all data written to it and by default verifies checksums when reading data. A separate checksum is created for every io.bytes.per.checksum bytes of data. The default is 512 bytes, and since a CRC-32 checksum is 4 bytes long, the storage overhead is less than 1%.

Datanodes are responsible for verifying the data they receive before storing the data and its checksum. This applies to data that they receive from clients and from other datanodes during replication. A client writing data sends it to a pipeline of datanodes and the last datanode in the pipeline verifies the checksum. If it detects an error, the client receives a ChecksumException, a subclass of IOException, which it should handle in an application-specific manner, by retrying the operation, for example.

When clients read data from datanodes, they verify checksums as well, comparing them with the ones stored at the datanode. Each datanode keeps a persistent log of checksum verifications, so it knows the last time each of its blocks was verified. When a client successfully verifies a block, it tells the datanode, which updates its log. Keeping statistics such as these is valuable in detecting bad disks. Aside from block verification on client reads, each datanode runs a DataBlockScanner in a background thread that periodically verifies all the blocks stored on the datanode. This is to guard against corruption due to "bit rot" in the physical storage media. See Datanode block scanner for details on how to access the scanner reports.

Hadoop I/O: compression, serialization

Since HDFS stores replicas of blocks, it can “heal” corrupted blocks by copying one of the good replicas to produce a new, uncorrupt replica. The way this works is that if a client detects an error when reading a block, it reports the bad block and the datanode it was trying to read from to the namenode before throwing a ChecksumException. The namenode marks the block replica as corrupt, so it doesn’t direct clients to it, or try to copy this replica to another datanode. It then schedules a copy of the block to be replicated on another datanode, so its replication factor is back at the expected level. Once this has happened, the corrupt replica is deleted.

LocalFileSystem
The Hadoop LocalFileSystem performs client-side checksumming. This means that when you write a file called filename, the filesystem client transparently creates a hidden file, .filename.crc, in the same directory containing the checksums for each chunk of the file. Like HDFS, the chunk size is controlled by the io.bytes.per.checksum property, which defaults to 512 bytes. The chunk size is stored as metadata in the .crc file, so the file can be read back correctly even if the setting for the chunk size has changed. Checksums are verified when the file is read, and if an error is detected, LocalFileSystem throws a ChecksumException.

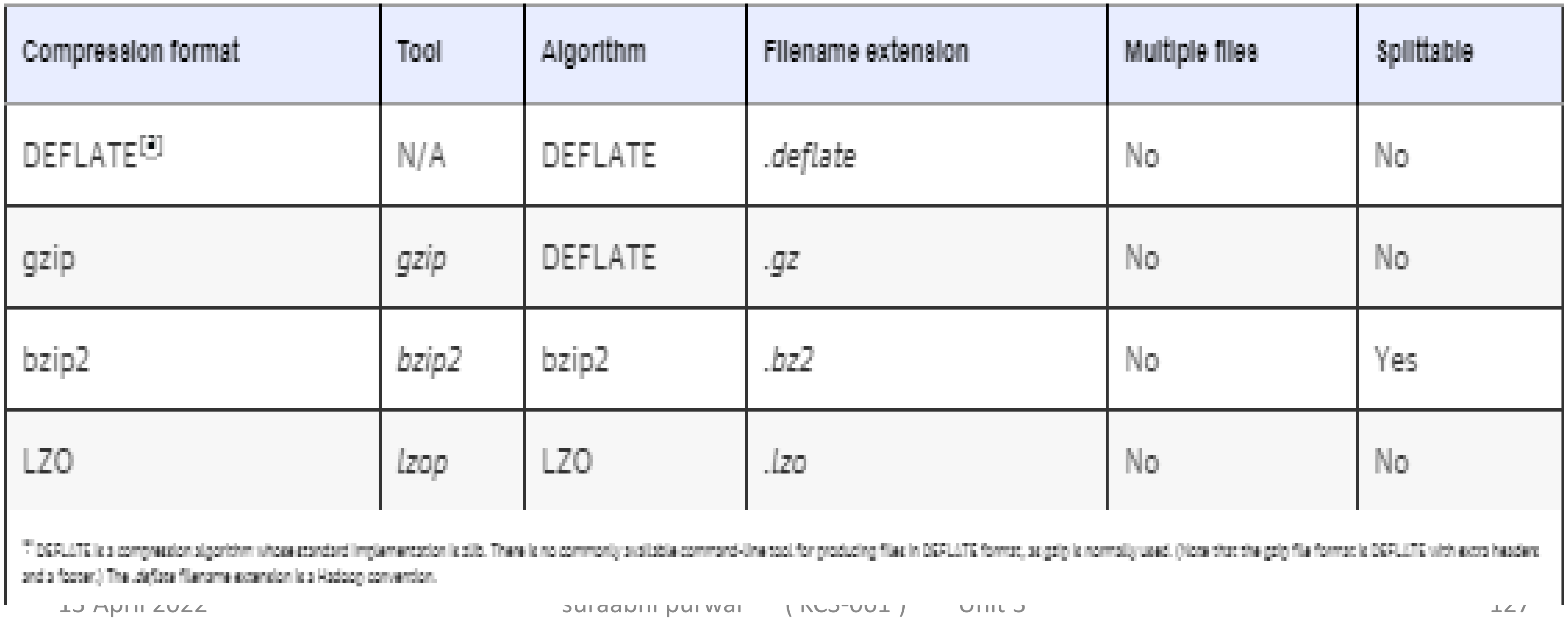
Hadoop I/O: compression, serialization

Compression

File compression brings two major benefits: it reduces the space needed to store files, and it speeds up data transfer across the network, or to or from disk. When dealing with large volumes of data, both of these savings can be significant, so it pays to carefully consider how to use compression in Hadoop.

There are many different compression formats, tools and algorithms, each with different characteristics. Table lists some of the more common ones that can be used with Hadoop.

Table 4-1. A summary of compression formats



Compression format	Tool	Algorithm	Filename extension	Multiple files	Splittable
DEFLATE[*]	N/A	DEFLATE	*.deflate*	No	No
gzip	*gzip*	DEFLATE	*.gz*	No	No
bzip2	*bzip2*	bzip2	*.bz2*	No	Yes
LZO	*lzop*	LZO	*.lzo*	No	No

[*] DEFLATE is a compression algorithm whose standard implementation is zlib. There is no commonly available command-line tool for producing files in DEFLATE format, as gzip is normally used. (Note that the gzip file format is DEFLATE with extra headers and a footer.) The .deflate filename extension is a Hadoop convention.

NIET Greater Noida GET FUTURE READY

Hadoop I/O: compression, serialization

All compression algorithms exhibit a space/time trade-off: faster compression and decompression speeds usually come at the expense of smaller space savings. All of the tools listed in Table 4-1 give some control over this trade-off at compression time by offering nine different options: –1 means optimize for speed and -9 means optimize for space. For example, the following command creates a compressed file file.gz using the fastest compression method:

```
gzip -1 file
```

The different tools have very different compression characteristics. Gzip is a general-purpose compressor, and sits in the middle of the space/time trade-off. Bzip2 compresses more effectively than gzip, but is slower. Bzip2's decompression speed is faster than its compression speed, but it is still slower than the other formats. LZO, on the other hand, optimizes for speed: it is faster than gzip (or any other compression or decompression tool[32]), but compresses slightly less effectively.

The "Splittable" column in Table 4-1 indicates whether the compression format supports splitting; that is, whether you can seek to any point in the stream and start reading from some point further on. Splittable compression formats are especially suitable for MapReduce; see Compression and Input Splits for further discussion.

Hadoop I/O: compression, serialization

Serialization

Serialization is the process of turning structured objects into a byte stream for transmission over a network or for writing to persistent storage. Deserialization is the reverse process of turning a byte stream back into a series of structured objects.

Serialization appears in two quite distinct areas of distributed data processing: for interprocess communication and for persistent storage.

In Hadoop, interprocess communication between nodes in the system is implemented using remote procedure calls (RPCs). The RPC protocol uses serialization to render the message into a binary stream to be sent to the remote node, which then deserializes the binary stream into the original message. In general, it is desirable that an RPC serialization format is:

Compact

A compact format makes the best use of network bandwidth, which is the most scarce resource in a data center.

Fast

Interprocess communication forms the backbone for a distributed system, so it is essential that there is as little performance overhead as possible for the serialization and deserialization process.

Hadoop I/O: compression, serialization

Extensible
Protocols change over time to meet new requirements, so it should be straightforward to evolve the protocol in a controlled manner for clients and servers. For example, it should be possible to add a new argument to a method call, and have the new servers accept messages in the old format (without the new argument) from old clients.

Interoperable
For some systems, it is desirable to be able to support clients that are written in different languages to the server, so the format needs to be designed to make this possible.

Avro and file-based data structures

Objective:

- **In this topic** we focus on Avro is a preferred tool to serialize data in Hadoop. Avro has a schema-based system. ... Avro serializes the data into a compact binary format, which can be deserialized by any application. Avro uses JSON format to declare the data structures.

Recap:

- Revision of ecosystem of Hadoop.

Avro and file-based data structures

To transfer data over a network or for its persistent storage, you need to serialize the data. Prior to the serialization APIs provided by Java and Hadoop, we have a special utility, called Avro, a schema-based serialization technique.

This tutorial teaches you how to serialize and deserialize the data using Avro. Avro provides libraries for various programming languages. In this tutorial, we demonstrate the examples using Java library.
What is Avro?
Apache Avro is a language-neutral data serialization system. It was developed by Doug Cutting, the father of Hadoop. Since Hadoop writable classes lack language portability, Avro becomes quite helpful, as it deals with data formats that can be processed by multiple languages. Avro is a preferred tool to serialize data in Hadoop.

Avro has a schema-based system. A language-independent schema is associated with its read and write operations. Avro serializes the data which has a built-in schema. Avro serializes the data into a compact binary format, which can be deserialized by any application.

Avro uses JSON format to declare the data structures. Presently, it supports languages such as Java, C, C++, C#, Python, and Ruby.

Avro and file-based data structures

Avro Schemas

Avro depends heavily on its schema. It allows every data to be written with no prior knowledge of the schema. It serializes fast and the resulting serialized data is lesser in size. Schema is stored along with the Avro data in a file for any further processing.

In RPC, the client and the server exchange schemas during the connection. This exchange helps in the communication between same named fields, missing fields, extra fields, etc.

Avro schemas are defined with JSON that simplifies its implementation in languages with JSON libraries.

Like Avro, there are other serialization mechanisms in Hadoop such as Sequence Files, Protocol Buffers, and Thrift.

Avro and file-based data structures

Comparison with Thrift and Protocol Buffers

Thrift and Protocol Buffers are the most competent libraries with Avro. Avro differs from these frameworks in the following ways −

Avro supports both dynamic and static types as per the requirement. Protocol Buffers and Thrift use Interface Definition Languages (IDLs) to specify schemas and their types. These IDLs are used to generate code for serialization and deserialization.

Avro is built in the Hadoop ecosystem. Thrift and Protocol Buffers are not built in Hadoop ecosystem.

Unlike Thrift and Protocol Buffer, Avro's schema definition is in JSON and not in any proprietary IDL.

Property	Avro	Thrift & Protocol Buffer
Dynamic schema	Yes	No
Built into Hadoop	Yes	No
Schema in JSON	Yes	No
No need to compile	Yes	No
No need to declare IDs	Yes	No
Bleeding edge	Yes	No

Avro and file-based data structures

Features of Avro

Listed below are some of the prominent features of Avro −

Avro is a language-neutral data serialization system.

It can be processed by many languages (currently C, C++, C#, Java, Python, and Ruby).

Avro creates binary structured format that is both compressible and splittable. Hence it can be efficiently used as the input to Hadoop MapReduce jobs.

Avro provides rich data structures. For example, you can create a record that contains an array, an enumerated type, and a sub record. These datatypes can be created in any language, can be processed in Hadoop, and the results can be fed to a third language.

Avro schemas defined in JSON, facilitate implementation in the languages that already have JSON libraries.

Avro creates a self-describing file named Avro Data File, in which it stores data along with its schema in the metadata section.

Avro is also used in Remote Procedure Calls (RPCs). During RPC, client and server exchange schemas in the connection handshake.

Avro and file-based data structures

General Working of Avro

To use Avro, you need to follow the given workflow −

Step 1 − Create schemas. Here you need to design Avro schema according to your data.

Step 2 − Read the schemas into your program. It is done in two ways −

By Generating a Class Corresponding to Schema − Compile the schema using Avro. This generates a class file corresponding to the schema

By Using Parsers Library − You can directly read the schema using parsers library.

Step 3 − Serialize the data using the serialization API provided for Avro, which is found in the package org.apache.avro.specific.

Step 4 − Deserialize the data using deserialization API provided for Avro, which is found in the package org.apache.avro.specific.

Security in Hadoop

Objective:

- **In this topic** we focus on Hadoop supports encryption at the disk, file system, database, and application levels. In core Hadoop technology the HFDS has directories called encryption zones. When data is written to Hadoop it is automatically encrypted (with a user-selected algorithm) and assigned to an encryption zone.

Recap:

- Revision of Ethics and compliance in Big Data.

Security in Hadoop

Hadoop is the software framework for storing and processing vast amounts of data. In this article, we will study Hadoop Security. The article first explains the reason for Hadoop Security. Then we will explore the Hadoop Security, 3 A’s in Hadoop Security, and how Hadoop achieves security. The article describes the Kerberos, transparent encryption in HDFS, and HDFS file and directory permission, which solves HDFS security issues. The article also enlists some Hadoop ecosystem components for monitoring and managing Hadoop Security.

Why Hadoop Security?

The goal of designing Hadoop is to manage large amounts of data in a trusted environment, so security was not a significant concern. But with the rise of the digital universe and the adoption of Hadoop in almost every sector like businesses, finance, health care, military, education, government, etc., security becomes the major concern.

The previous Hadoop implementation lacked security features because the built-in security and available options are inconsistent among release versions, which affect many sectors like multiple business sectors, health, medical departments, national security, and military, etc. It became obvious that there should be a mechanism that ensures Hadoop Security.

So, Hadoop security is the major leap that Hadoop Framework needs to take.

Security in Hadoop

What to do about Hadoop Security?

The security in Hadoop requires many of the same approaches seen in the traditional data management system. These include 3 A's of security.

Let us first see what these 3 A's says:

Authentication: It means "Who am I/prove it?".

Authorization: It means, "What can I do?"

Auditing: It means, "What did I do?"

Data Protection: It means, "How can I encrypt the data at rest and over the wire?".

Authentication: It is the first stage that strongly authenticates the user to prove their identities. In authentication, user credentials like UserId, password are authenticated. Authentication ensures that the user who is seeking to perform an operation is the one who he claims to be and thus trustable.

Authorization: It is the second stage that defines what individual users can do after they have been authenticated. Authorization controls what a particular user can do to a specific file. It provides permission to the user whether he can access the data or not.

Auditing: Auditing is the process of keeping track of what an authenticated, authorized user did once he gets access to the cluster. It records all the activity of the authenticated user, including what data was accessed, added, changed, and what analysis occurred by the user from the period when he login to the cluster.

Data Protection: It refers to the use of techniques like encryption and data masking for preventing sensitive data access by unauthorized users and applications.

Security in Hadoop

Introduction to Hadoop Security

Around 2009, Hadoop’s security was designed and implemented and had been stabilizing since then. In 2010, the security feature added in Hadoop with the following two fundamental goals:

Preventing unauthorized access to the files stored in HDFS.

Not exceeding high cost while achieving authorization.

Hadoop Security thus refers to the process that provides authentication, authorization, auditing, and secure the Hadoop data storage unit by offering an inviolable wall of security against any cyber threat.

Administering Hadoop, HDFS monitoring & maintenance

Objective:

- **In this topic** we focus on MapReduce - Hadoop Administration
- HDFS administration includes monitoring the HDFS file structure, locations, and the updated files.
- MapReduce administration includes monitoring the list of applications, configuration of nodes, application status, etc

Recap:

- Revision of Mapper & Reducer architecture.

Administering Hadoop, HDFS monitoring & maintenance

Hadoop administration which includes both HDFS and MapReduce administration.

HDFS administration includes monitoring the HDFS file structure, locations, and the updated files.

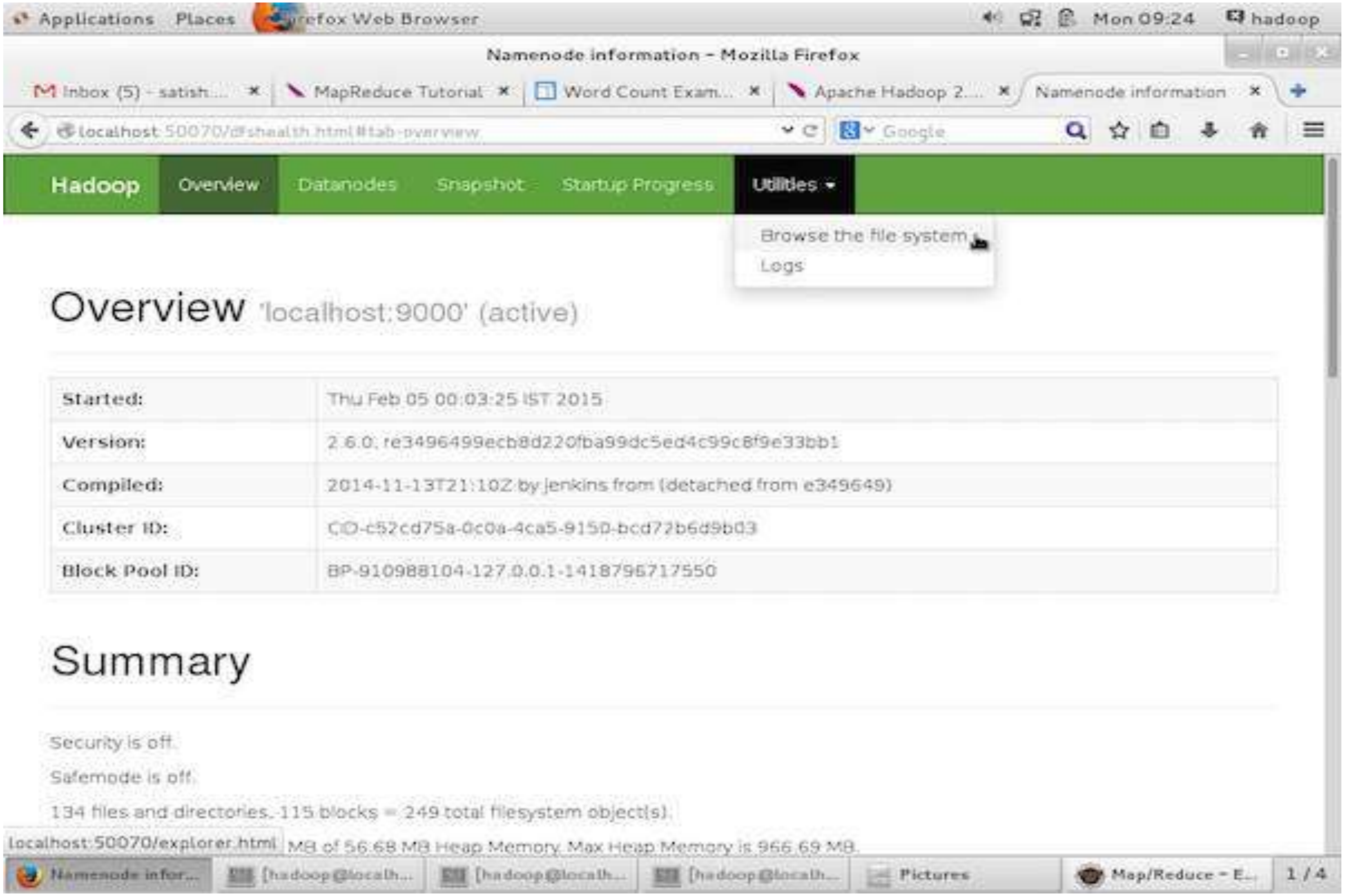
MapReduce administration includes monitoring the list of applications, configuration of nodes, application status, etc.

HDFS Monitoring
HDFS (Hadoop Distributed File System) contains the user directories, input files, and output files. Use the MapReduce commands, put and get, for storing and retrieving.

After starting the Hadoop framework (daemons) by passing the command “start-all.sh” on “/$HADOOP_HOME/sbin”, pass the following URL to the browser “http://localhost:50070”. You should see the following screen on your browser.

The following screenshot shows how to browse the browse HDFS

Administering Hadoop, HDFS monitoring & maintenance



Overview 'localhost:9000' (active)

Started:	Thu Feb 05 00:03:25 IST 2015
Version:	2.6.0, re3496499ecb8d220fba99dc5ed4c99c8f9e33bb1
Compiled:	2014-11-13T21:10Z by jenkins from (detached from e349649)
Cluster ID:	CID-c52cd75a-0c0a-4ca5-9150-bcd72b6d9b03
Block Pool ID:	BP-910988104-127.0.0.1-1418796717550

Summary

Security is off.

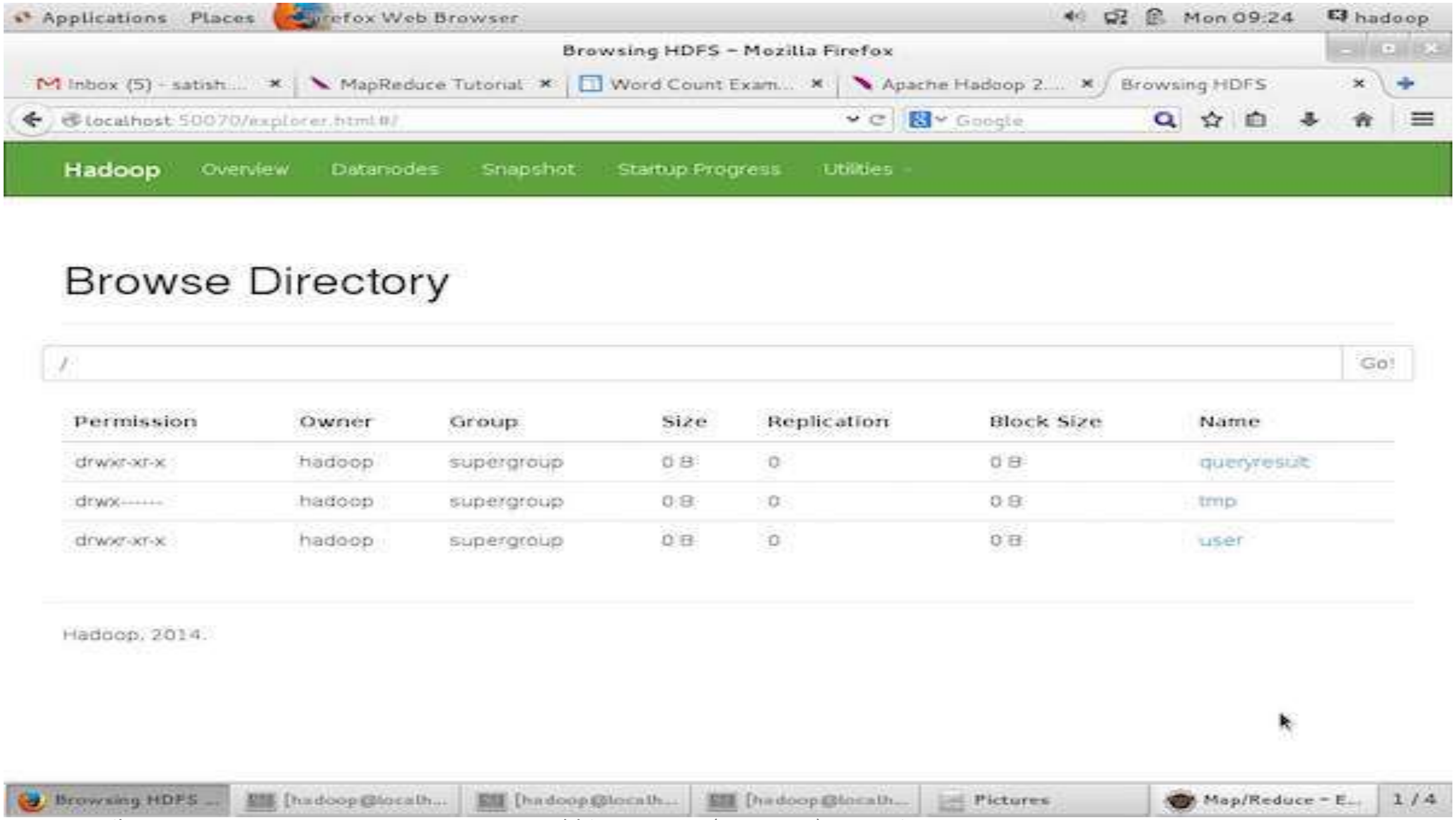
Safemode is off.

134 files and directories, 115 blocks = 249 total filesystem object(s).

MB of 56.68 MB Heap Memory. Max Heap Memory is 966.69 MB.

Administering Hadoop, HDFS monitoring & maintenance

The following screenshot show the file structure of HDFS. It shows the files in the "/user/hadoop" directory.


Applications Places Firefox Web Browser Mon 09:24 hadoop
Browsing HDFS – Mozilla Firefox
Inbox (5) - satish... MapReduce Tutorial Word Count Exam... Apache Hadoop 2.... Browsing HDFS
localhost:50070/explorer.html#/
Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/ Go!

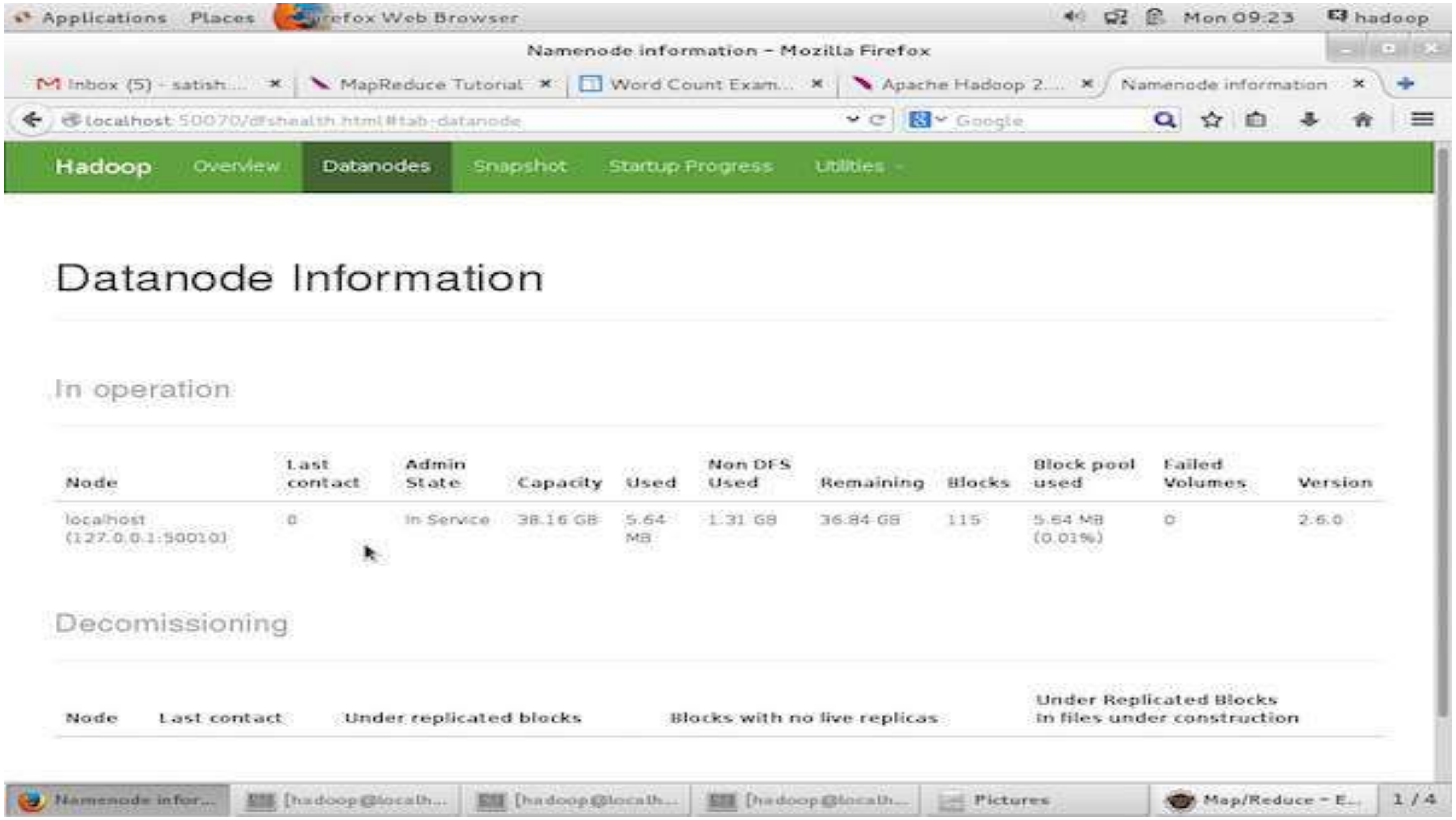
Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	queryresult
drwx------	hadoop	supergroup	0 B	0	0 B	tmp
drwxr-xr-x	hadoop	supergroup	0 B	0	0 B	user

Hadoop, 2014.

Browsing HDFS ... [hadoop@localh... [hadoop@localh... [hadoop@localh... Pictures Map/Reduce - E... 1 / 4

Administering Hadoop, HDFS monitoring & maintenance

The following screenshot shows the Datanode information in a cluster. Here you can find one node with its configurations and capacities.


Applications Places Firefox Web Browser Mon 09:23 hadoop
Namenode information - Mozilla Firefox
localhost:50070/dfshealth.html#tab-datanode
Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
localhost (127.0.0.1:50010)	0	In Service	38.16 GB	5.64 MB	1.31 GB	36.84 GB	115	5.64 MB (0.01%)	0	2.6.0

Decomissioning

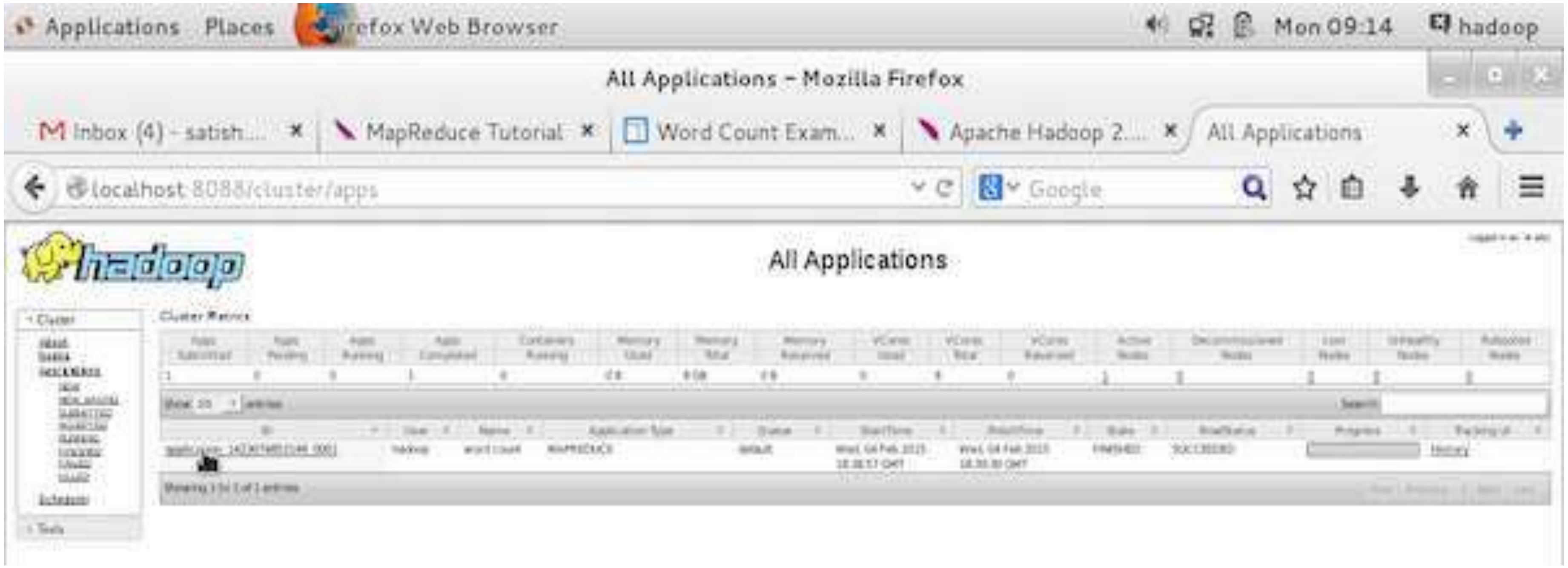
Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction

Administering Hadoop, HDFS monitoring & maintenance

MapReduce Job Monitoring

A MapReduce application is a collection of jobs (Map job, Combiner, Partitioner, and Reduce job). It is mandatory to monitor and maintain the following −

Configuration of datanode where the application is suitable.
The number of datanodes and resources used per application.
To monitor all these things, it is imperative that we should have a user interface. After starting the Hadoop framework by passing the command "start-all.sh" on "/$HADOOP_HOME/sbin", pass the following URL to the browser "http://localhost:8080". You should see the following screen on your browser.



Administering Hadoop, HDFS monitoring & maintenance

In the above screenshot, the hand pointer is on the application ID. Just click on it to find the following screen on your browser. It describes the following −

On which user the current application is running

The application name

Type of that application

Current status, Final status

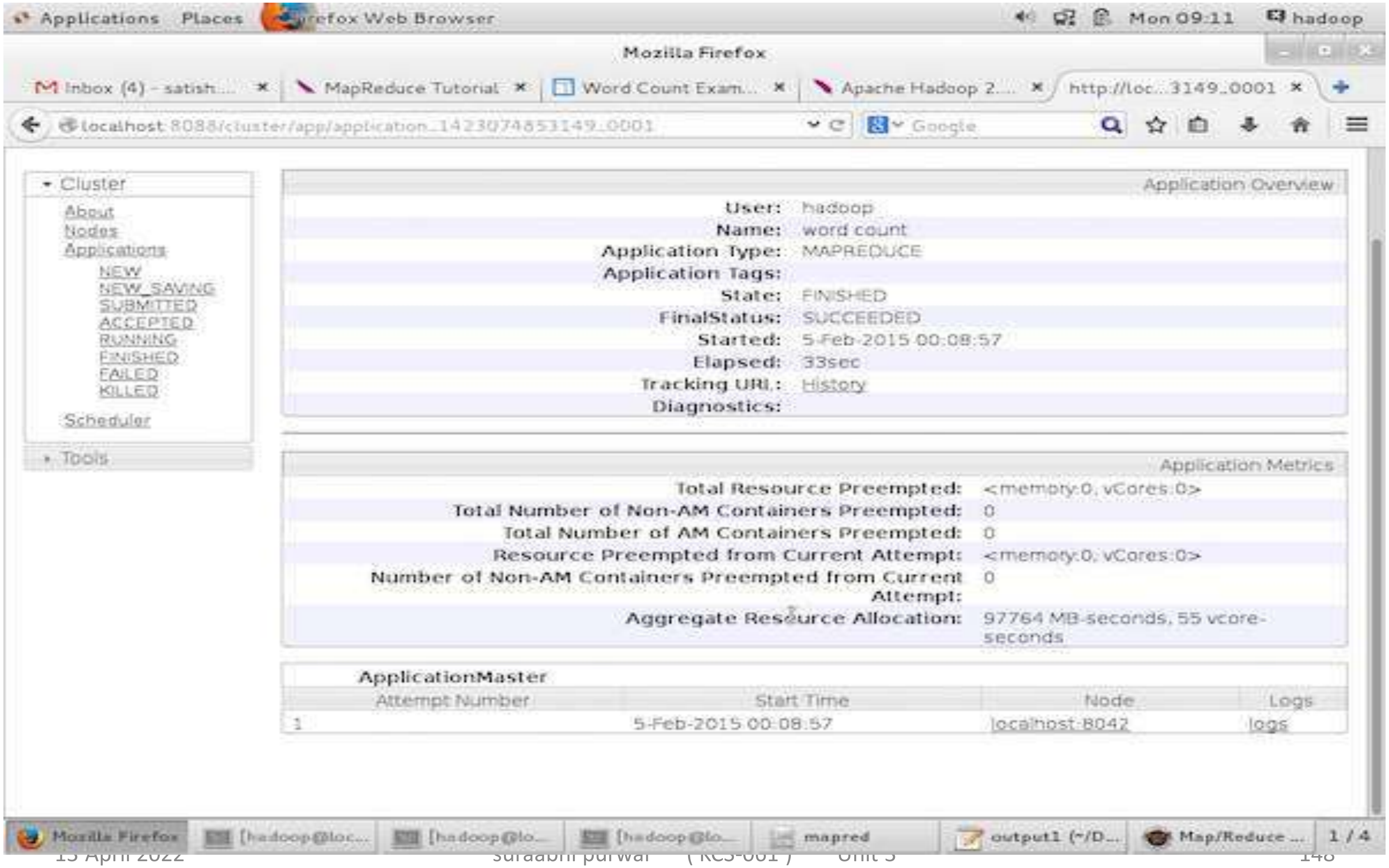
Application started time, elapsed (completed time), if it is complete at the time of monitoring

The history of this application, i.e., log information

And finally, the node information, i.e., the nodes that participated in running the application.

Administering Hadoop, HDFS monitoring & maintenance

The following screenshot shows the details of a particular application −



Application Overview

User:	hadoop
Name:	word count
Application Type:	MAPREDUCE
Application Tags:	
State:	FINISHED
FinalStatus:	SUCCEEDED
Started:	5-Feb-2015 00:08:57
Elapsed:	33sec
Tracking URL:	History
Diagnostics:	

Application Metrics

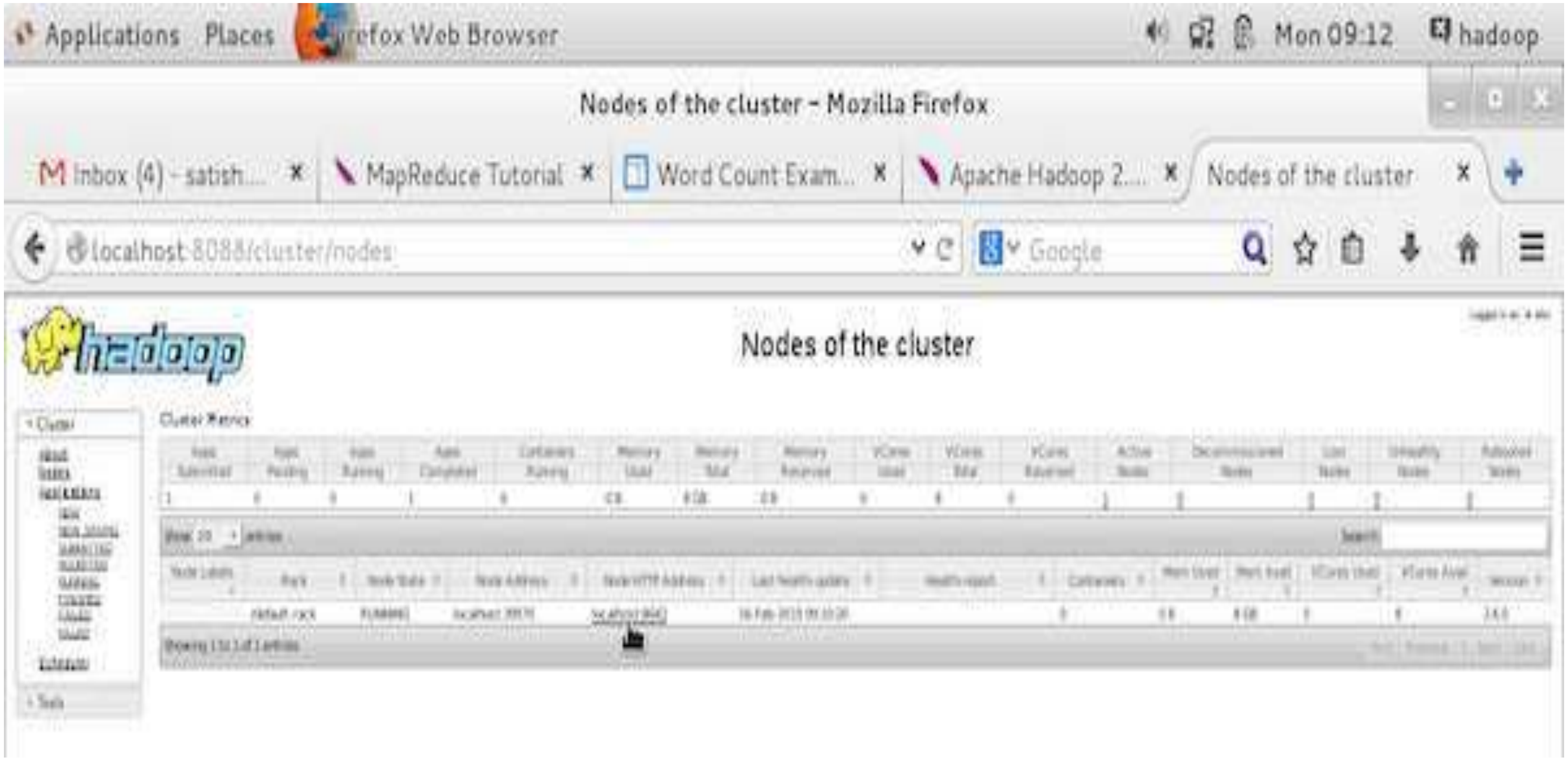
Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	97764 MB-seconds, 55 vcore-seconds

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	5-Feb-2015 00:08:57	localhost:8042	logs

Administering Hadoop, HDFS monitoring & maintenance

The following screenshot describes the currently running nodes information. Here, the screenshot contains only one node. A hand pointer shows the localhost address of the running node.



Hadoop benchmarks

Objective:

- **In this topic** we focus on TeraSort Benchmark is used to test both, MapReduce and HDFS by sorting some amount of data as quickly as possible in order to measure the capabilities of distributing and mapreducing files in cluster. This benchmark consists of 3 components: TeraGen - generates random data. TeraSort - does the sorting using MapReduce.

Recap:

- Revision of Hadoop Architecture.

Hadoop benchmarks

NNThroughputBenchmark

Overview

NNThroughputBenchmark, as its name indicates, is a name-node throughput benchmark, which runs a series of client threads on a single node against a name-node. If no name-node is configured, it will firstly start a name-node in the same process (standalone mode), in which case each client repetitively performs the same operation by directly calling the respective name-node methods. Otherwise, the benchmark will perform the operations against a remote name-node via client protocol RPCs (remote mode). Either way, all clients are running locally in a single process rather than remotely across different nodes. The reason is to avoid communication overhead caused by RPC connections and serialization, and thus reveal the upper bound of pure name-node performance.

The benchmark first generates inputs for each thread so that the input generation overhead does not effect the resulting statistics. The number of operations performed by threads is practically the same. Precisely, the difference between the number of operations performed by any two threads does not exceed 1. Then the benchmark executes the specified number of operations using the specified number of threads and outputs the resulting stats by measuring the number of operations performed by the name-node per second.

Hadoop in the cloud

Objective:

- **In this topic** we focus on Apache Hadoop software is an open source framework that allows for the distributed storage and processing of large datasets across clusters of computers using simple programming models. ... In this way, Hadoop can efficiently store and process large datasets ranging in size from gigabytes to petabytes of data.

Recap:

- Revision of Cloud Architecture.

Hadoop in the cloud

What Does Hadoop in the Cloud Mean?

"Hadoop in the cloud" means: it is running Hadoop clusters on resources offered by a cloud provider. This practice is normally compared with running Hadoop clusters on your own hardware, called on-premises clusters or "on-prem."

After all, a cloud instance is supposed to act almost exactly like an ordinary server you connect to remotely, with root access, and some number of CPU cores, and some amount of disk space, and so on. Once instances are networked together properly and made accessible, you can imagine that they are running in a regular data center, as opposed to a cloud provider's own data center. This illusion is intentional, so that working in a cloud provider feels familiar, and your skills still apply.

That doesn't mean there's nothing new to learn, or that the abstraction is complete. A cloud provider does not do everything for you; there are many choices and a variety of provider features to understand and consider, so that you can build not only a functioning system, but a functioning system of Hadoop clusters. Cloud providers also include features that go beyond what you can do on-prem, and Hadoop clusters can benefit from those as well.

Mature Hadoop clusters rarely run in isolation. Supporting resources around them manage data flow in and out and host specialized tools, applications backed by the clusters, and non-Hadoop servers, among other things. The supporting cast can also run in the cloud, or else dedicated networking features can help to bring them close.

Hadoop in the cloud

Reasons to Run Hadoop in the Cloud

Many concepts have been defined so far, but the core question has not yet been answered: Why run Hadoop clusters in the cloud at all? Here are just a few reasons:

Lack of space

Your organization may need Hadoop clusters, but you don't have anywhere to keep racks of physical servers, along with the necessary power and cooling.

Flexibility

Without physical servers to rack up or cables to run, it is much easier to reorganize instances, or expand or contract your footprint, for changing business needs. Everything is controlled through cloud provider APIs and web consoles. Changes can be scripted and put into effect manually or even automatically and dynamically based on current conditions.

New usage patterns

The flexibility of making changes in the cloud leads to new usage patterns that are otherwise impractical. For example, individuals can have their own instances, clusters, and even networks, without much managerial overhead. The overall budget for CPU cores in your cloud provider account can be concentrated in a set of large instances, a larger set of smaller instances, or some mixture, and can even change over time.

Hadoop in the cloud

Speed of change
It is much faster to launch new cloud instances or allocate new database servers than to purchase, unpack, rack, and configure physical computers. Similarly, unused resources in the cloud can be torn down swiftly, whereas unused hardware tends to linger wastefully.

Lower risk
How much on-prem hardware should you buy? If you don’t have enough, the entire business slows down. If you buy too much, you’ve wasted money and have idle hardware that continues to waste money. In the cloud, you can quickly and easily change how many resources you use, so there is little risk of undercommitment or overcommitment. What’s more, if some resource malfunctions, you don’t need to fix it; you can discard it and allocate a new one.

Focus
An organization using a cloud provider to rent resources, instead of spending time and effort on the logistics of purchasing and maintaining its own physical hardware and networks, is free to focus on its core competencies, like using Hadoop clusters to carry out its business. This is a compelling advantage for a tech startup, for example.

Hadoop in the cloud

Worldwide availability
The largest cloud providers have data centers around the world, ready for you from the start. You can use resources close to where you work, or close to where your customers are, for the best performance. You can set up redundant clusters, or even entire computing environments, in multiple data centers, so that if local problems occur in one data center, you can shift to working elsewhere.

Data storage requirements
If you have data that is required by law to be stored within specific geographic areas, you can keep it in clusters that are hosted in data centers in those areas.

Cloud provider features
Each major cloud provider offers an ecosystem of features to support the core functions of computing, networking, and storage. To use those features most effectively, your clusters should run in the cloud provider as well.

Capacity
Few customers tax the infrastructure of a major cloud provider. You can establish large systems in the cloud that are not nearly as easy to put together, not to mention maintain, on-prem