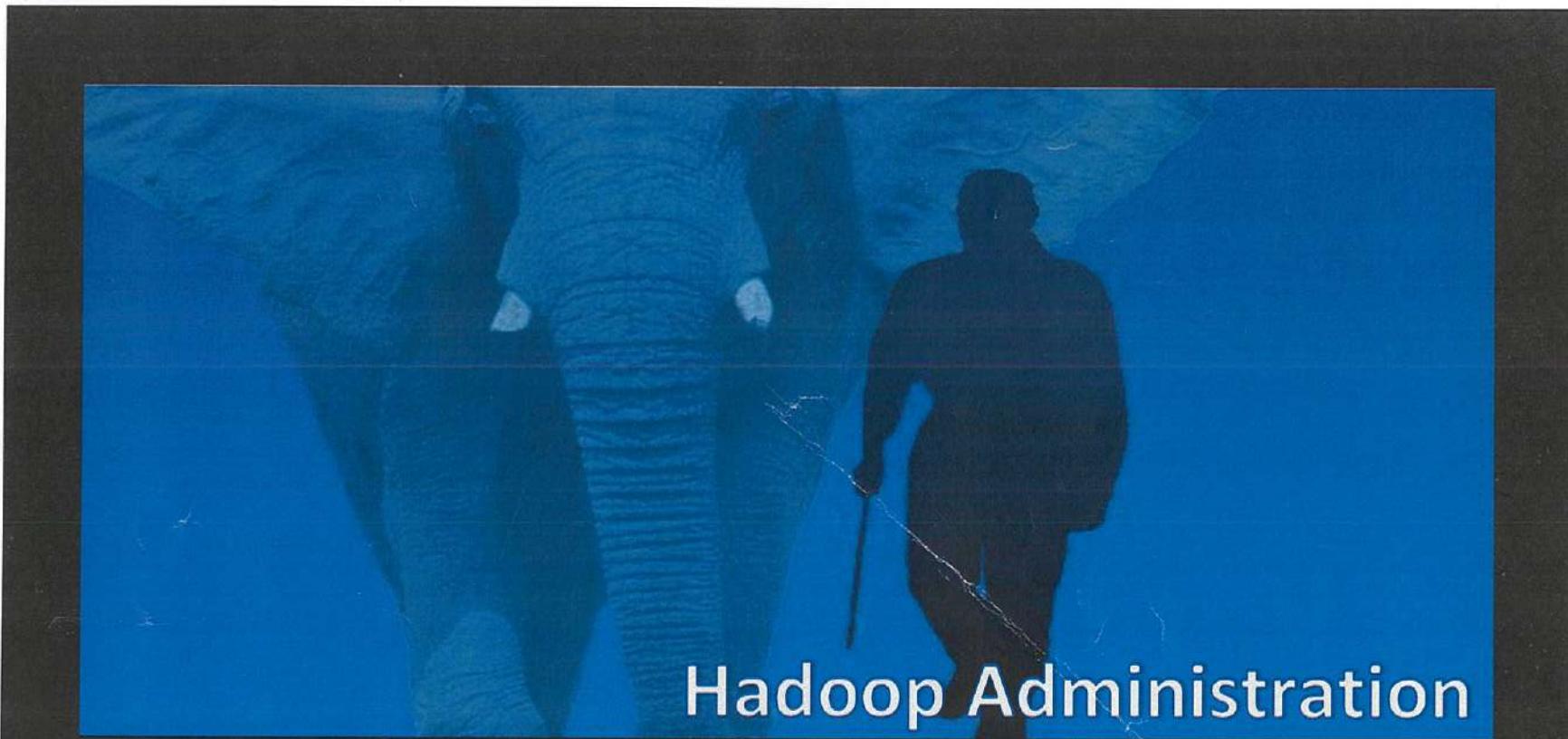


S. Vijay Kumar

8105 823 876

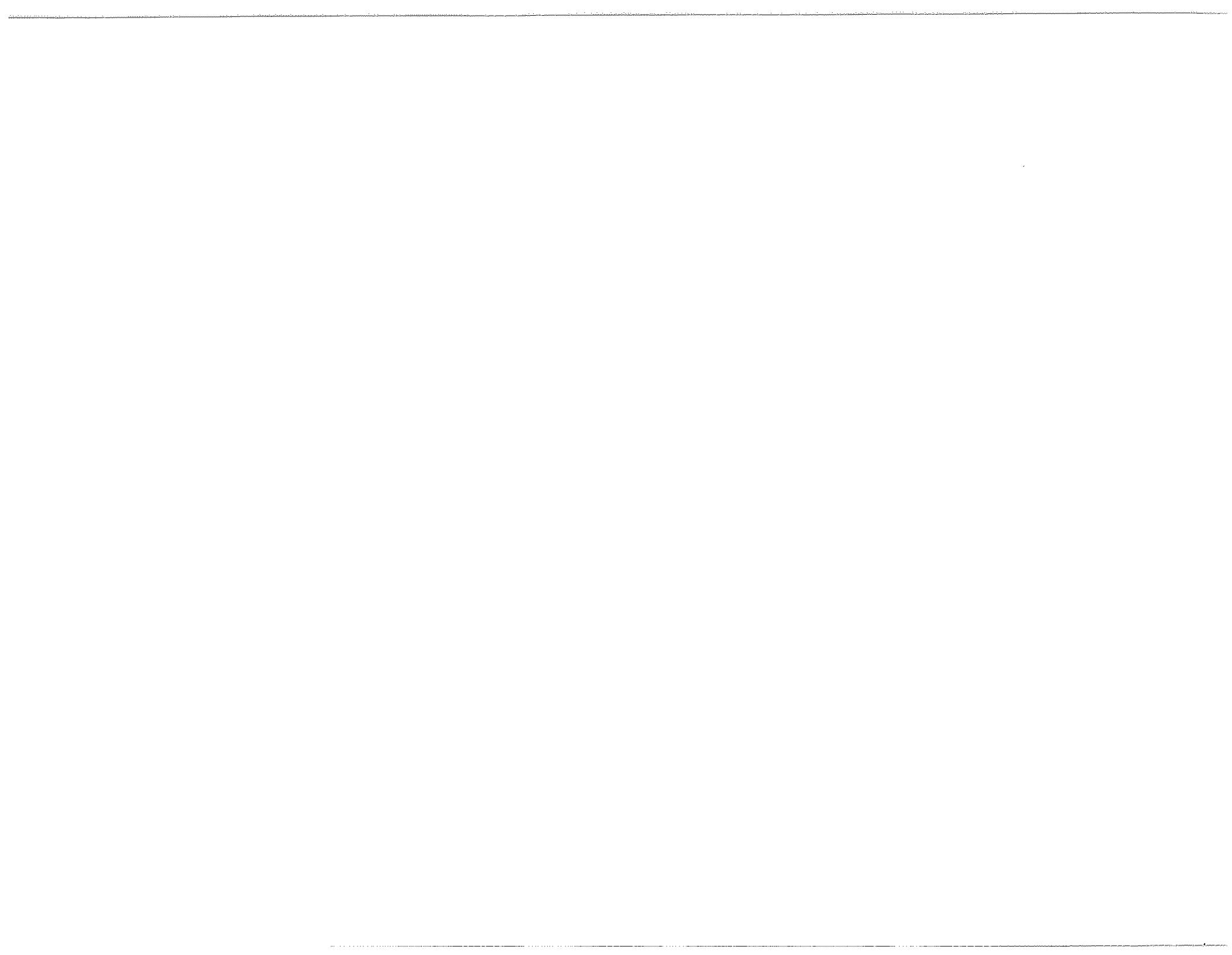


Training Version – 2.4

Email: narasimha.v.rao.b@gmail.com

Venkata Narasimha Rao B  
(+91 9342707000)

2<sup>nd</sup> Floor, Sri Krishna Sagar Hotel Building, Anandhapura,  
T.C. Palya Main Road, K.R. Puram, Bangalore – 560036.





## Table of Contents

	Page #
<b>What is Big Data?</b>	<b>11</b>
<b>Where this data is coming from?</b>	<b>12</b>
<b>What are Big Data use cases?</b>	<b>13</b>
<b>How Data is growing?</b>	<b>14</b>
<b>3 V's of Big Data</b>	<b>15</b>
<b>What do we need now?</b>	<b>16</b>
<b>What is Hadoop?</b>	<b>17</b>
<b>Sample Project Architecture</b>	<b>18</b>
<b>Hadoop Project Architecture (Administration Vs Development)</b>	<b>19</b>
<b>Sample Expense Report for a Startup Company</b>	<b>20</b>
<b>Hadoop Admin Prerequisites</b>	<b>21</b>
<b>What are we going to target?</b>	<b>22</b>
<b>Understanding of File System</b>	<b>24</b>
<b>Understanding INode Structure</b>	<b>25</b>
<b>Understanding of File write process on disk</b>	<b>26</b>
<b>What if my file is stored in non-continuous blocks?</b>	<b>28</b>
<b>What if file size is larger than disk?</b>	<b>29</b>
<b>What if we maintain Master FAT across multiple disks with larger block size?</b>	<b>30</b>
<b>Sample file storage on Master FAT</b>	<b>31</b>
<b>Understanding Distributed File System</b>	<b>32</b>
<b>What are the advantages with Distributed File System?</b>	<b>33</b>
<b>Hadoop Components</b>	<b>34</b>
<b>Hadoop Architecture</b>	<b>35</b>





## Table of Contents

	Page #
<b>Storage Layer: HDFS</b>	<b>36</b>
<b>How file is stored on HDFS?</b>	<b>38</b>
<b>HDFS: The Hadoop Distributed File System</b>	<b>40</b>
<b>HDFS Key Points</b>	<b>41</b>
<b>HDFS Features</b>	<b>42</b>
<b>HDFS Design Assumptions</b>	<b>43</b>
<b>Formatting the NameNode</b>	<b>44</b>
<b>DataNode Registration with NameNode</b>	<b>45</b>
<b>Communication between NameNode and DataNode</b>	<b>46</b>
<b>Communication between Client and NameNode</b>	<b>47</b>
<b>How metadata is maintained in Hadoop?</b>	<b>48</b>
<b>Metadata in NameNode's Memory</b>	<b>49</b>
<b>What is Block Report?</b>	<b>50</b>
<b>Checkpointing Mechanism</b>	<b>51</b>
<b>When Checkpoint will occur?</b>	<b>52</b>
<b>Secondary NameNode</b>	<b>53</b>
<b>How Checkpoint happens in the Cluster?</b>	<b>54</b>
<b>Installation Modes of Hadoop</b>	<b>56</b>
<b>Linux OS Flavors</b>	<b>57</b>
<b>Most commonly used Linux Commands in Hadoop Troubleshooting</b>	<b>58</b>
<b>Preparing a Node for Hadoop Cluster</b>	<b>63</b>
<b>Why Oracle JDK instead of Open JDK?</b>	<b>64</b>
<b>Hadoop Configuration Files</b>	<b>65</b>
<b>Hadoop 1.X Architecture</b>	<b>66</b>
<b>Limitations of Hadoop 1.X Storage Layer</b>	<b>67</b>





## Table of Contents

	Page #
<b>What is Edge Node?</b>	<b>68</b>
<b>Hadoop 2.X Architecture - NameNode HA (Using NFS Shared Filer)</b>	<b>71</b>
<b>Limitations of NameNode HA using NFS Shared Filer</b>	<b>72</b>
<b>ZooKeeper (ZK)</b>	<b>73</b>
<b>What are the types of ZNodes?</b>	<b>74</b>
<b>Why odd number of Nodes in ZK Ensemble?</b>	<b>75</b>
<b>Hadoop 2.X Architecture - NameNode HA (Using ZooKeeper &amp; NFS Shared Filer)</b>	<b>76</b>
<b>Journal Nodes (JN)</b>	<b>77</b>
<b>Hadoop 2.X Architecture - NameNode HA (Using Journal Nodes &amp; ZooKeeper)</b>	<b>78</b>
<b>How to change edits from NFS Shared Filer to Journal Nodes?</b>	<b>79</b>
<b>Advantages of NameNode HA with ZooKeeper &amp; Journal Nodes</b>	<b>80</b>
<b>Fencing</b>	<b>81</b>
<b>HDFS Federation</b>	<b>82</b>
<b>Hadoop 3.X Setup differences in Storage Layer</b>	<b>83</b>
<b>Hadoop 3.X Architecture</b>	<b>84</b>
<b>NameNode Startup Process</b>	<b>86</b>
<b>Hadoop Safemode</b>	<b>87</b>
<b>NameNode Heap Memory requirements to store metadata</b>	<b>88</b>
<b>Why HDFS Metadata to be maintained in RAM?</b>	<b>89</b>
<b>Anatomy of a File Write into HDFS</b>	<b>90</b>
<b>What if a DataNode in the pipeline is failed?</b>	<b>92</b>
<b>Dealing with multiple Disks in Hadoop</b>	<b>93</b>
<b>Checking HDFS Status</b>	<b>94</b>
<b>HDFS Block Replication Strategy (Rack Awareness)</b>	<b>95</b>
<b>Why Rack Awareness?</b>	<b>96</b>





## Table of Contents

	Page #
<b>Anatomy of a File Read from HDFS</b>	<b>97</b>
<b>Dealing with data corruption (Data Integrity)</b>	<b>99</b>
<b>Dealing with data corruption (Setting up Trash)</b>	<b>100</b>
<b>Space Reclamation</b>	<b>101</b>
<b>DataNode Failure, Heartbeats and Re-Replication</b>	<b>102</b>
<b>When DataNode marked as failed?</b>	<b>103</b>
<b>Hadoop Cluster Rebalancing</b>	<b>104</b>
<b>Change of Replication Factor</b>	<b>106</b>
<b>Change of Block Size</b>	<b>107</b>
<b>File Systems supported by Hadoop</b>	<b>108</b>
<b>Compression formats supported by Hadoop</b>	<b>110</b>
<b>How GZIP works:</b>	<b>111</b>
<b>How LZO works?</b>	<b>112</b>
<b>Comparison of Compression formats</b>	<b>113</b>
<b>CORE</b>	<b>115</b>
<b>Java Virtual Memory (JVM) Internals</b>	<b>116</b>
<b>Map Reduce (High Level Architecture)</b>	<b>117</b>
<b>Job Tracker (JT)</b>	<b>118</b>
<b>Task Tracker (TT)</b>	<b>119</b>
<b>How does MapReduce work?</b>	<b>120</b>
<b>MapReduce Pipeline</b>	<b>121</b>
<b>Relationship between Input Split and HDFS Block</b>	<b>122</b>
<b>Anatomy of MapReduce Job Submission</b>	<b>123</b>
<b>Running simple MR Jobs in Hadoop</b>	<b>124</b>
<b>How the tasks will be submitted to Task Trackers?</b>	<b>125</b>





## Table of Contents

	Page #
Choosing the number of Mappers & Reducers	126
Controlling maximum length of a record	127
Commissioning (Adding) Nodes to the Cluster	128
De-Commissioning (Removing) Nodes from the Cluster	129
Limitations of Job Tracker	130
Yet Another Resource Negotiator (YARN) Components	131
YARN Architecture	132
Resource Manager's Responsibilities	133
Job Submission in YARN	134
Memory Allocation in YARN	136
Advantages of YARN framework	137
Job/Application ID, Task ID, and Task Attempt IDs	139
Counters in Hadoop	140
Job History	142
Logging & Log Files	143
Hadoop Daemon Logs (for Administrators)	144
Job Statistics Logs (for Users)	145
Job Logs	146
Standard Error Logs (for Users)	147
System Logs (for Users)	148
Configuring the Logging-Related Parameters in YARN	149
Speculative Execution	150
Failures in Job execution	151
Limitations of Single Resource Manager	155
Resource Manager High Availability	156



## Table of Contents

	Page #
<b>Blacklisting the NodeManagers</b>	<b>159</b>
<b>Hadoop File Formats</b>	<b>160</b>
<b>Hadoop Distributions</b>	<b>166</b>
<b>Different Types of Cloudera Distribution</b>	<b>167</b>
<b>Different Types of Hortonworks Distributions</b>	<b>168</b>
<b>Different Types of MapR Distributions</b>	<b>169</b>
<b>Cloudera Architecture</b>	<b>170</b>
<b>Ambari Architecture</b>	<b>171</b>
<b>MapR Control System (MCS) Architecture</b>	<b>172</b>
<b>Cloudera Vs Hortonworks Vs MapR</b>	<b>173</b>
<b>Sqoop</b>	<b>177</b>
<b>Sqoop Architecture</b>	<b>178</b>
<b>Sqoop Features</b>	<b>179</b>
<b>Performance Tuning in SQOOP</b>	<b>180</b>
<b>Realtime issues from SQOOP</b>	<b>181</b>
<b>Pig</b>	<b>182</b>
<b>Performance Tuning in PIG</b>	<b>183</b>
<b>Realtime issues from PIG</b>	<b>184</b>
<b>Hive</b>	<b>185</b>
<b>Hive Architecture</b>	<b>186</b>
<b>How HIVE works?</b>	<b>188</b>
<b>Hive Tables</b>	<b>189</b>
<b>Hive Table Partitions</b>	<b>190</b>
<b>Hive Table Bucketing</b>	<b>191</b>
<b>Hiveserver2 (HS2)</b>	<b>192</b>





## Table of Contents

	Page #
Hive Metastore	193
Hive Table Locks	194
Security on HIVE	195
Security on HIVE with Apache Sentry	196
Where / When to use HIVE?	197
Performance Tuning in Hive	198
Realtime issues from Hive	200
HBase	201
HBase Architecture	202
HBase Components	203
HBase Auto Sharding	206
HBase Performance Tuning	207
Realtime issues from HBase	208
Flume	209
Flume Architecture	210
Kafka	212
Kafka Architecture	213
Kafka Components	214
Kafka workflow	215
Realtime issues from Kafka	217
Storm	218
Storm Architecture	219
Storm Components	220
Storm Process Flow	222
Spark	223





## Table of Contents

	Page #
<b>Spark Architecture</b>	<b>224</b>
<b>Spark Components</b>	<b>225</b>
<b>Spark Deployment Modes</b>	<b>226</b>
<b>Resilient Distributed Datasets (RDD)</b>	<b>227</b>
<b>Advantages of Spark over native MapReduce Programming</b>	<b>228</b>
<b>Oozie</b>	<b>229</b>
<b>Oozie Architecture</b>	<b>230</b>
<b>Types of Oozie Jobs</b>	<b>231</b>
<b>Realtime issues from Oozie</b>	<b>232</b>
<b>Sample Real-time Project Architecture with Batch Processing</b>	<b>233</b>
<b>Sample Real-time Project Architecture with Realtime Processing</b>	<b>234</b>
<b>Sample Real-time Cluster</b>	<b>235</b>
<b>Hadoop Ecosystem Components</b>	<b>236</b>
<b>Components that has High Availability in Hadoop Eco System</b>	<b>237</b>
<b>Hadoop Ecosystem Components with latest stable versions</b>	<b>238</b>
<b>Cluster Planning &amp; Scaling</b>	<b>239</b>
<b>CPU</b>	<b>240</b>
<b>RAM</b>	<b>241</b>
<b>Hard Disk</b>	<b>242</b>
<b>Network</b>	<b>243</b>
<b>Commonly used Node Configuration in Hadoop</b>	<b>244</b>
<b>Plan your Cluster growth</b>	<b>245</b>
<b>Hadoop Cluster Upgrade</b>	<b>246</b>
<b>Types of Upgrade</b>	<b>247</b>
<b>Cloudera Hadoop Cluster Upgrade</b>	<b>248</b>





## Table of Contents

	Page #
Hortonworks Hadoop Cluster Upgrade	249
Hadoop Distcp	250
Hadoop Cluster Backup	251
Data Corruption on Disk	252
Metadata Failure in CM / Ambari / Hive / Oozie	253
Rack Failure	254
Disaster Recovery	255
HBase Data Backup & Recovery	256
HDFS Snapshots	258
HDFS Snapshot Commands	259
OS Patching	260
Manual Patching	261
Configuration Management Tools	262
Monitoring Hadoop Clusters	263
Nagios Mail Alert Sample	264
Ganglia Stats	265
Hadoop Security	267
Types of Hadoop Security	268
HDFS Access Control Lists (ACL)	269
What is Kerberos?	271
Kerberos Terminology	272
How Kerberos works?	273
Ranger	275
Ranger Architecture	276
Ranger Components	277





## Table of Contents

	Page #
<b>Knox Gateway</b>	<b>278</b>
<b>Knox Gateway Architecture</b>	<b>279</b>
<b>What is SSL, TLS &amp; HTTPS?</b>	<b>280</b>
<b>How does SSL work?</b>	<b>281</b>
<b>Common Sources of Problems</b>	<b>282</b>
<b>Operating System &amp; Network Tuning</b>	<b>284</b>
<b>Data Disk Scaling</b>	<b>287</b>
<b>Applying HDFS Quotas</b>	<b>288</b>
<b>Submitting Jobs to Queues</b>	<b>290</b>
<b>Hadoop Job Schedulers</b>	<b>291</b>
<b>FIFO Scheduler</b>	<b>292</b>
<b>Fair Scheduler</b>	<b>293</b>
<b>Capacity Scheduler</b>	<b>294</b>
<b>YARN Resource Allocation Models</b>	<b>295</b>
<b>Performance Tuning of a MapReduce Job</b>	<b>297</b>
<b>Performance Tuning of YARN Job</b>	<b>298</b>
<b>Hadoop Admin Responsibilities</b>	<b>302</b>





## What is Big Data?

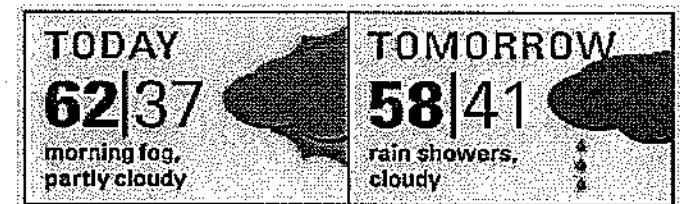
Data sets that are *too large* to store and process by using existing traditional systems.





## Where this data is coming from?

- Machine Data – Real-time data from Sensors
- Transactional Data – Records, Payments & Invoices
- Social Media Data – Likes, Tweets & Comments

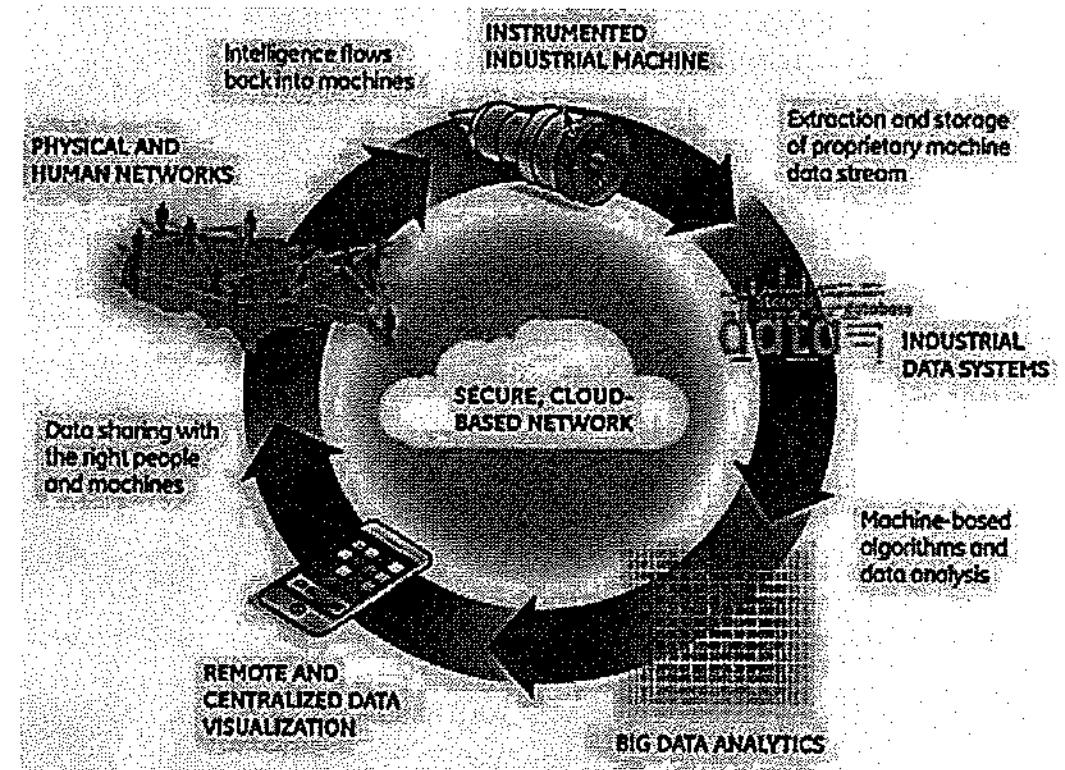


Symbol	Open	High	Low	Close	Contract	Last	2nd	H1	L0	P5	Low	High
3960	6526	6526	6526	6526	2005.C2000	3050		3050	3050	4702		
3958	6522	6522	6522	6522	2002.C1600	4700	4700	4700	4700	4837		
3972	6774	6774	6774	6774	200212.C3000	2300	2300	2300	2300	4978		
3955	6422	6422	6422	6422	2002.C1750	6300R					5122	
3952	6528	6528	6528	6528	2003.C1200	2650	3050	3050	2650	5267		
	18+	266-	266-	266-	2002.P1200	120		120	120	5415		
	180	238	238	238	2002.C1750	6300R					5564	
3950	13104	13104	13104	13104	2005.C1600	6000		6000	6000	5715		
3977	13102	13102	13102	13102	2002.C1750	7900		7900	7900	5868	5250	5250
3966	13208	13208	13208	13208	2001.C1200	6300		6300	6300	6023		



## What are Big Data use cases?

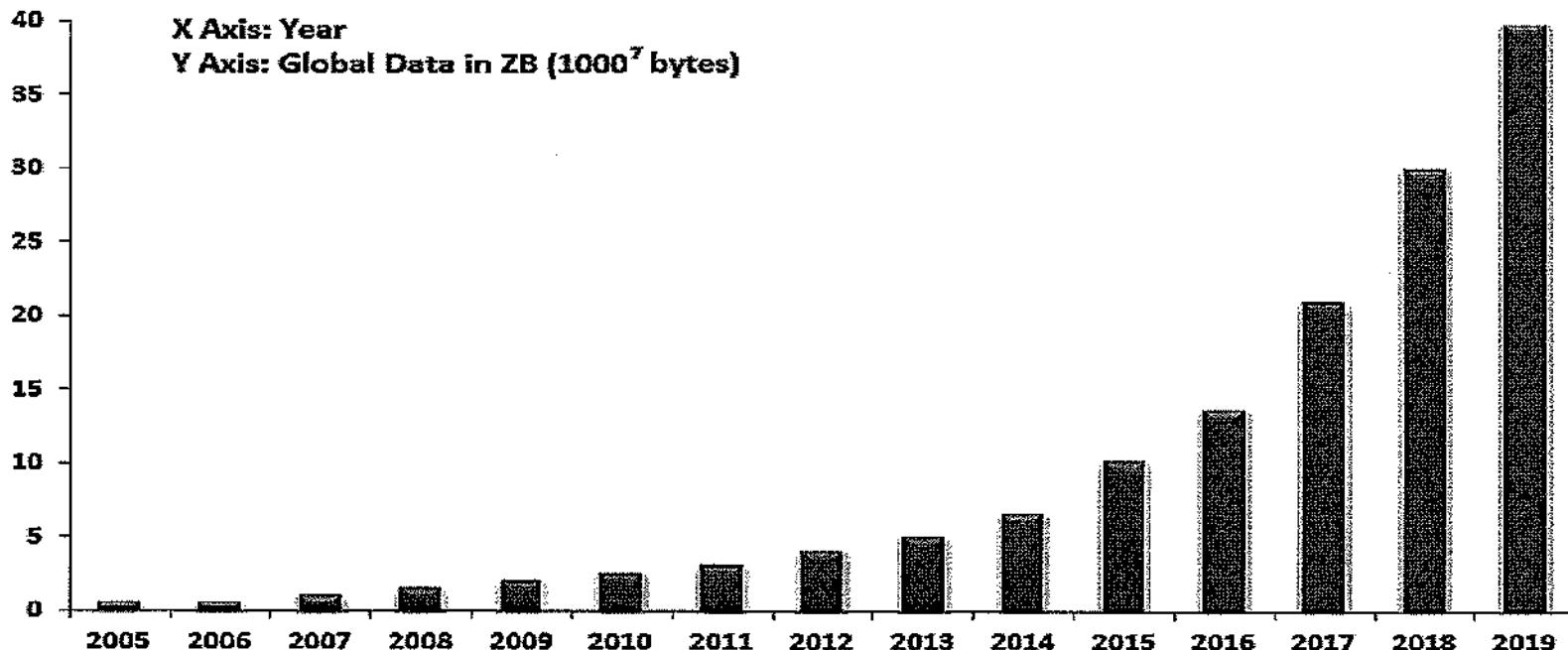
- Healthcare
- Aviation
- Research
- UIDAI - Aadhar Project
- Predictive & Behavioral Analysis
- Banking
- Stock Exchange
- Social Media
- Server Logs
- E-Mail & Text Messages



## How Data is growing?

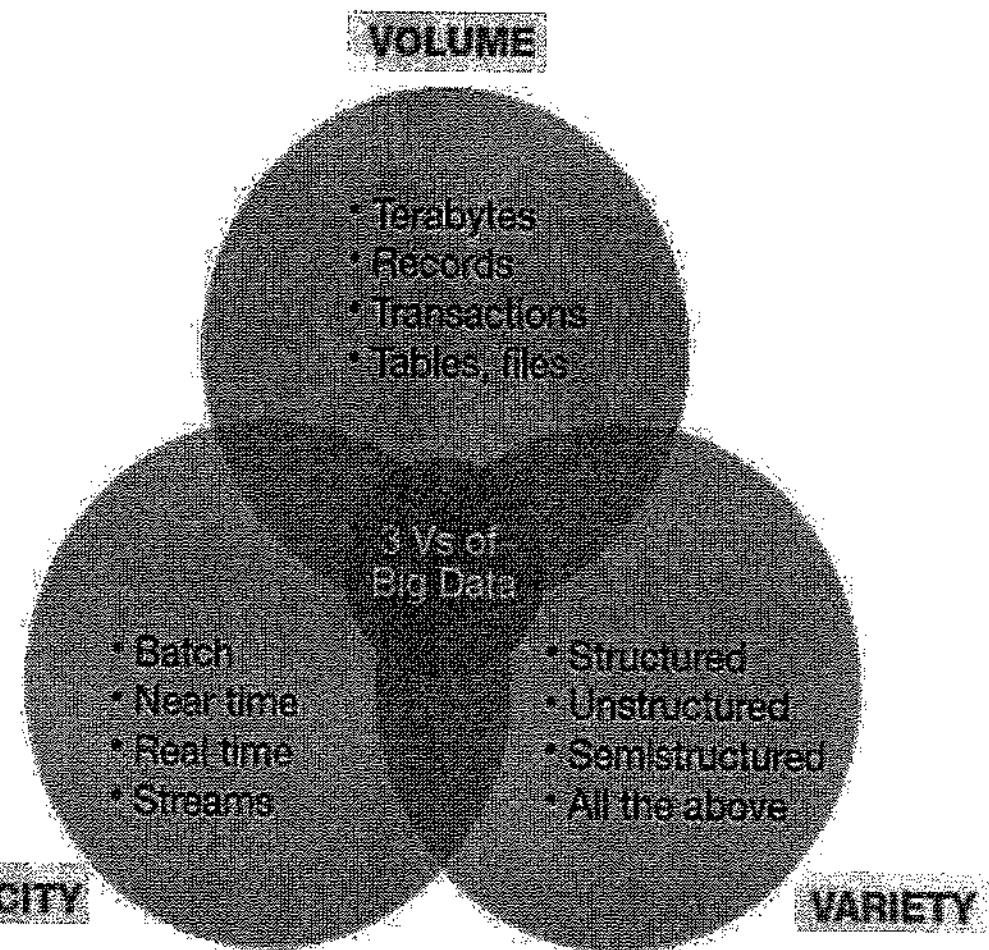
For example, *every day*

- Twitter processes **500 million** messages
- Amazon S3 storage adds more than **1 billion** objects
- Facebook users generates **4.5 billion** comments and “Likes”
- **90% of data in the world has been generated in last 3-4 years**





## 3 V's of Big Data





## What do we need now?

We need a platform, which could address all these issues / challenges.





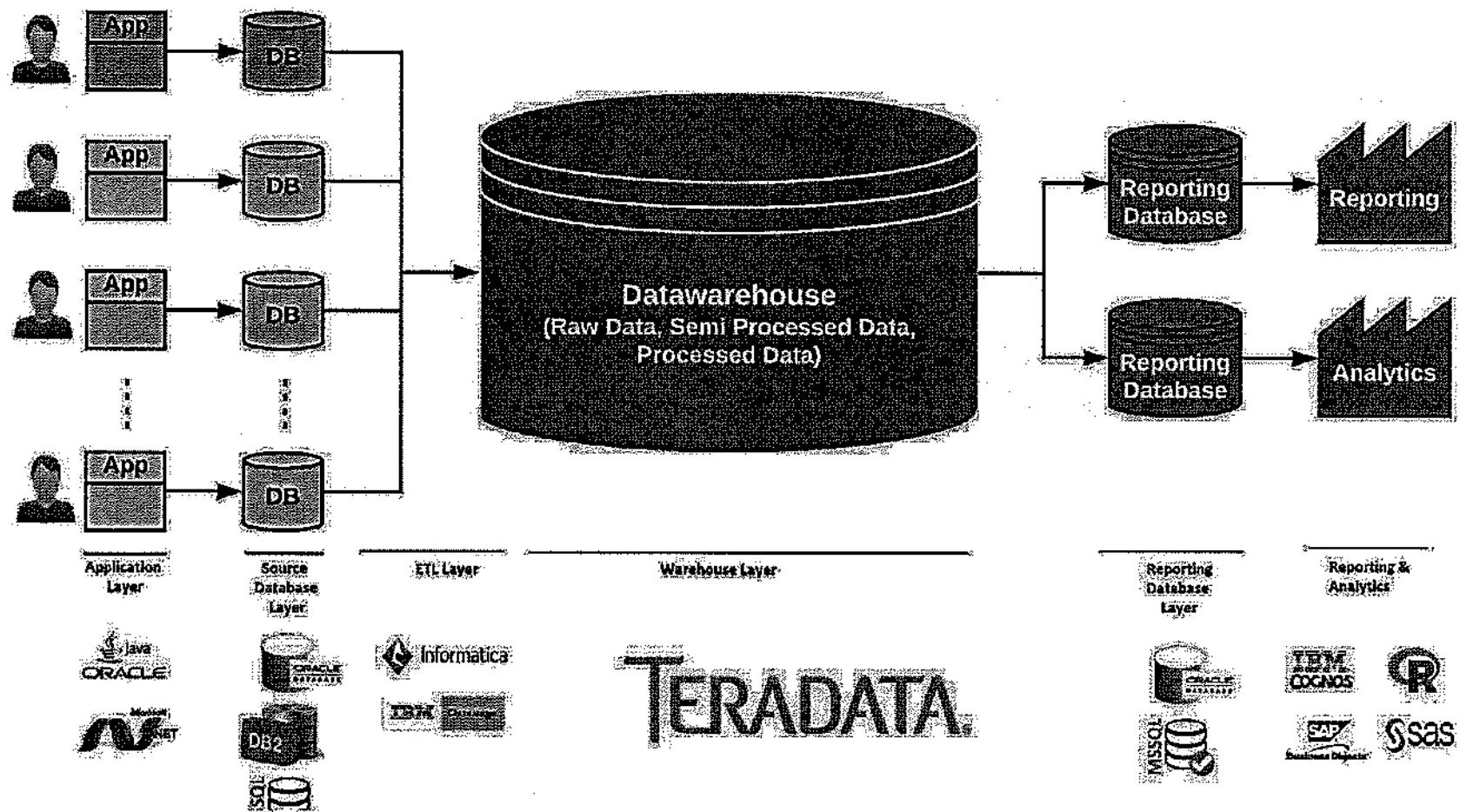
## What is Hadoop?

Hadoop is an *Open Source Framework* for the distributed processing of large data across Clusters of commodity computers using a simple programming model.





## Sample Project Architecture





## Hadoop Project Architecture (Administration Vs Development)





## Sample Expense Report for a Startup Company

	Vendor	Licensed Technologies		Open Source Technologies on Cloud	
		Monthly	Half-Yearly	Monthly	Half-Yearly
Office Space	---	2,00,000	12,00,000	---	---
Hardware	DELL Servers	2,00,00,000	2,00,00,000	2,50,000 (Rent)	15,00,000
Frontend	Oracle JAVA	2,00,000	12,00,000	Open JDK	---
Backend	Oracle DB	10,00,000	60,00,000	File System	---
Employees		10,00,000	60,00,000	10,00,000	60,00,000
Total			3,44,00,000		75,00,000





## Hadoop Admin Prerequisites:

- General operational expertise such as good troubleshooting skills, understanding of system's capacity, bottlenecks, basics of *CPU, Memory, Storage and Network*.
- Good knowledge of *Linux Commands* as Hadoop runs on Linux
- Very little *JAVA (optional)*
- Shell, Python or any other Scripting Knowledge (*optional*)
- Hadoop Skills
  - *Planning & Setting up new Hadoop Cluster*
  - *Configuring High Availability for critical components of Hadoop Cluster*
  - *Deploying and Monitoring of Ecosystem Tools on Hadoop Cluster*
  - *Enabling Security on Hadoop Cluster*
  - *User, Space & Resource Management in Hadoop Cluster*
  - *Keeping track of all jobs running on Hadoop Cluster*
  - *Monitoring critical parts of Hadoop Cluster*
  - *Commissioning and Decommissioning of Nodes to/from Hadoop Cluster*
  - *Performance tuning of Hadoop Cluster Components and Ecosystem Tools*
  - *Backup & Disaster Recovery of Hadoop Cluster's Data*
  - *Patching & Cluster upgrades*



## What are we going to target?

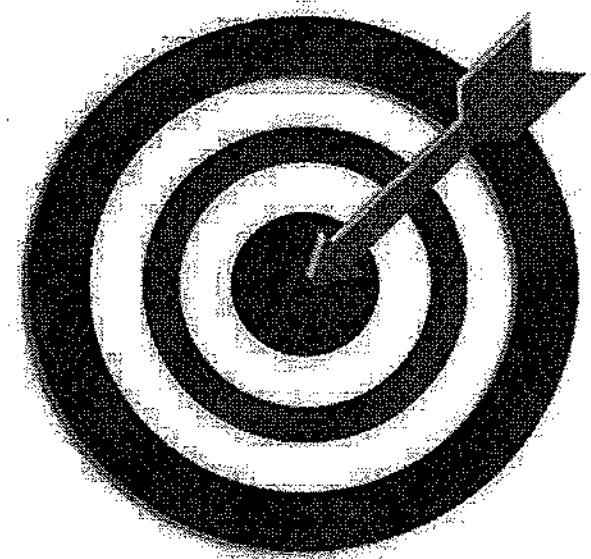
- Big Data and Introduction to Apache Hadoop
- Hadoop Components (HDFS & MapReduce) & Internals
- Hadoop 1x Installation (Distribution Mode) & Internals
- Hadoop 2x Installation with ZooKeeper & NFS Shared Filer
- Hadoop 2x Installation with ZooKeeper & Journal Nodes
- Hadoop 3x Installation with ZooKeeper & Journal Nodes
- NameNode & Resource Manager High Availability
- Cloudera Installation & Walkthrough
- Hortonworks Installation & Walkthrough
- MapR Installation & Walkthrough
- Brief & Installation of Hadoop Ecosystem Tools
  - Sqoop
  - HBase
  - Oozie
  - Pig
  - Flume
  - Kafka
  - Hive
  - Storm
  - Spark
- Cluster Planning





## What are we going to target? (Cont'd)

- Hadoop Cluster Upgrades
- Brief on Cluster Backup & Restore
- Hadoop Snapshots & Mirrors
- Brief on OS Patching
- Brief on Hadoop Patching
- Brief on Configuration Management Tools
- Monitoring of Hadoop Clusters
- Managing Users, Groups, Use Cases & Quotas
- Hadoop Security
  - LDAP
  - Hadoop ACLs
  - Kerberos
  - SSL, TLS & HTTPS
  - Ranger & Knox
- Hadoop Cluster Performance Tuning
- Real Time Q & A
  - Day to Day Activities
  - Roles and Responsibilities
  - Frequently occurring issues

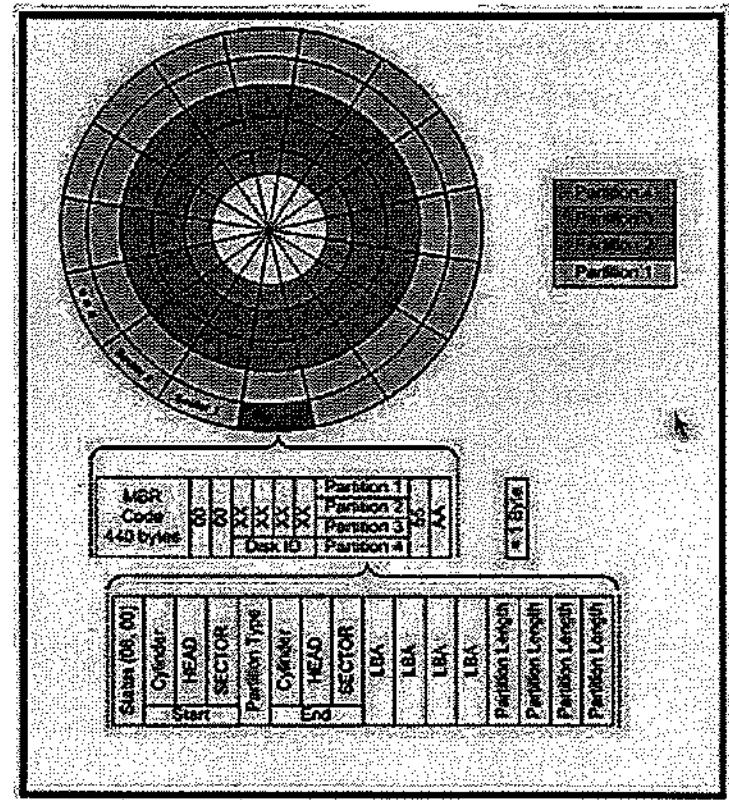




## **Understanding of File System**

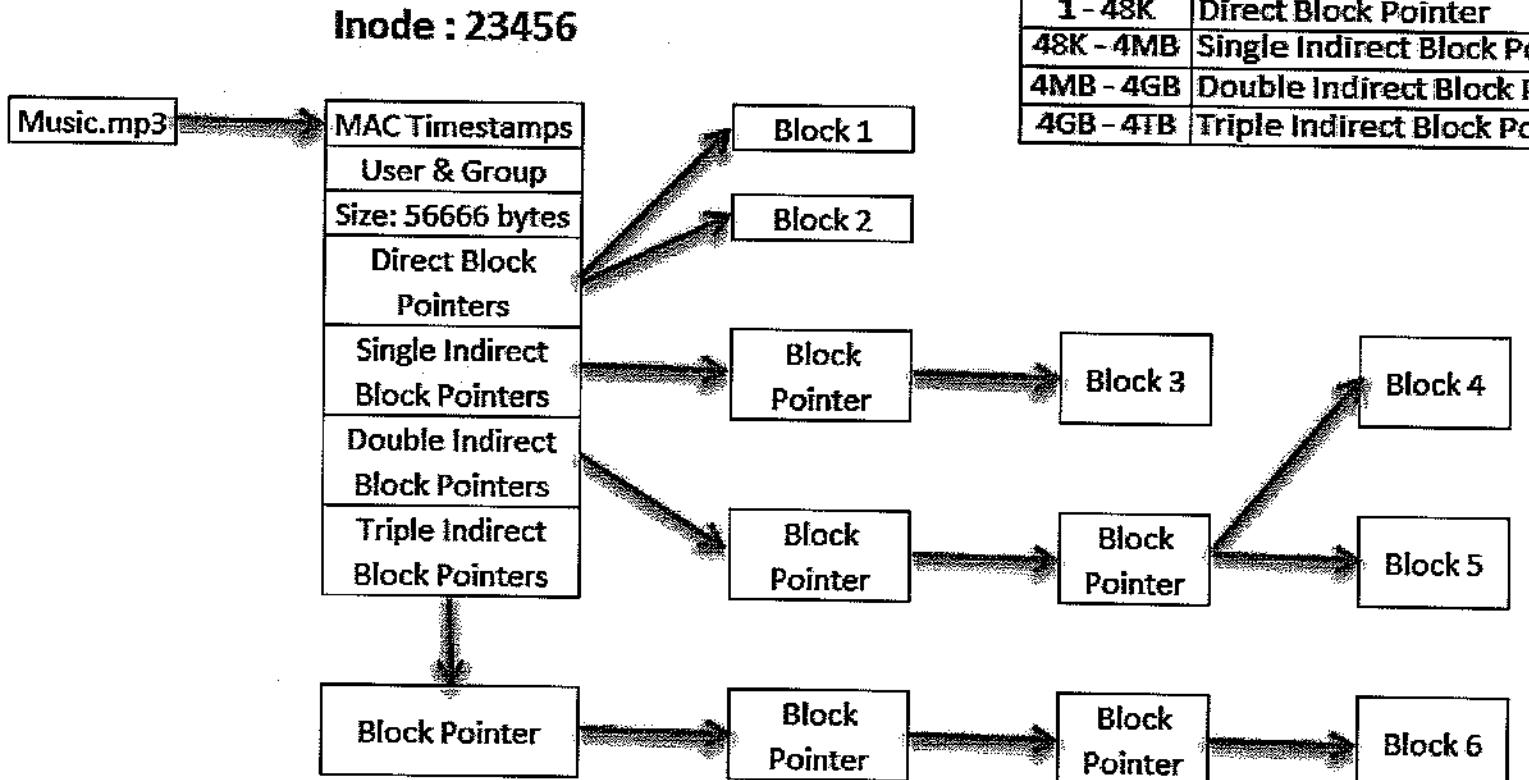
**Traditionally, data and metadata of a file system will sit on the same disk.**

- MBR** – Master Boot Record
  - FAT** – File Allocation Table
  - INodes** – File Data Structures or Pointers



## Understanding INode Structure

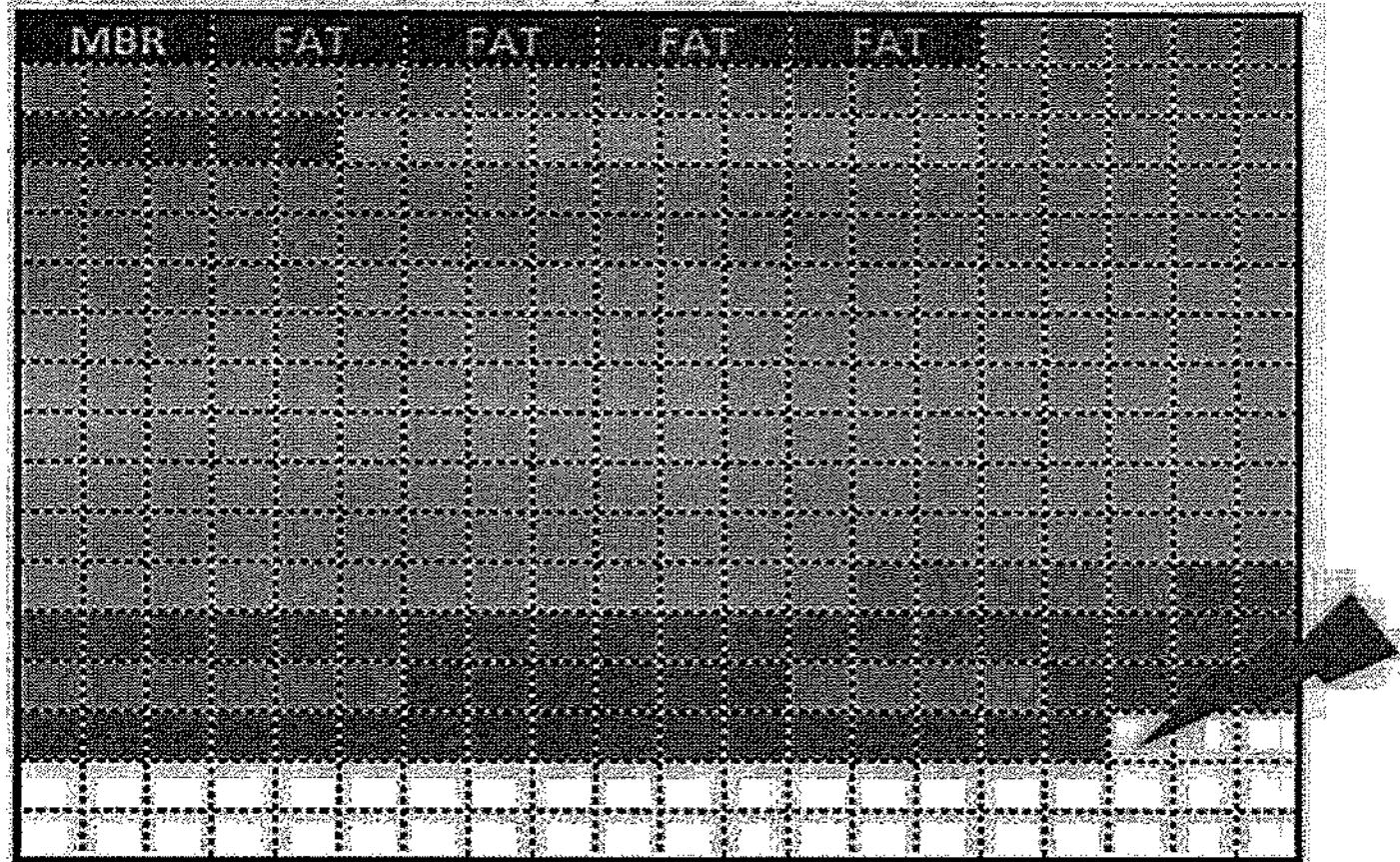
### INodes – File Data Structures or Pointers





## Understanding of File write process on disk

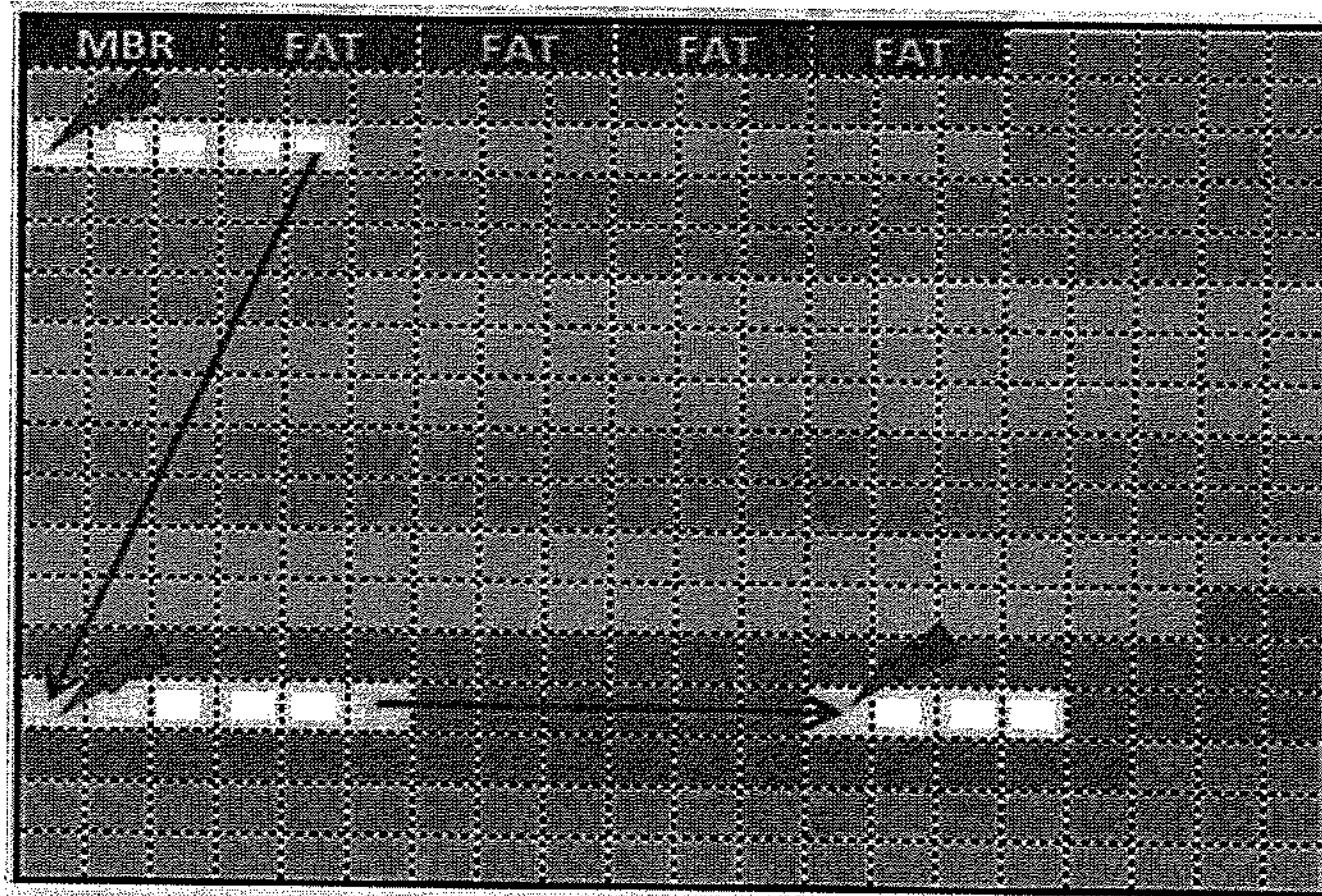
Unix File System : 4KB Blocks





## Understanding of File write process on disk (Cont'd)

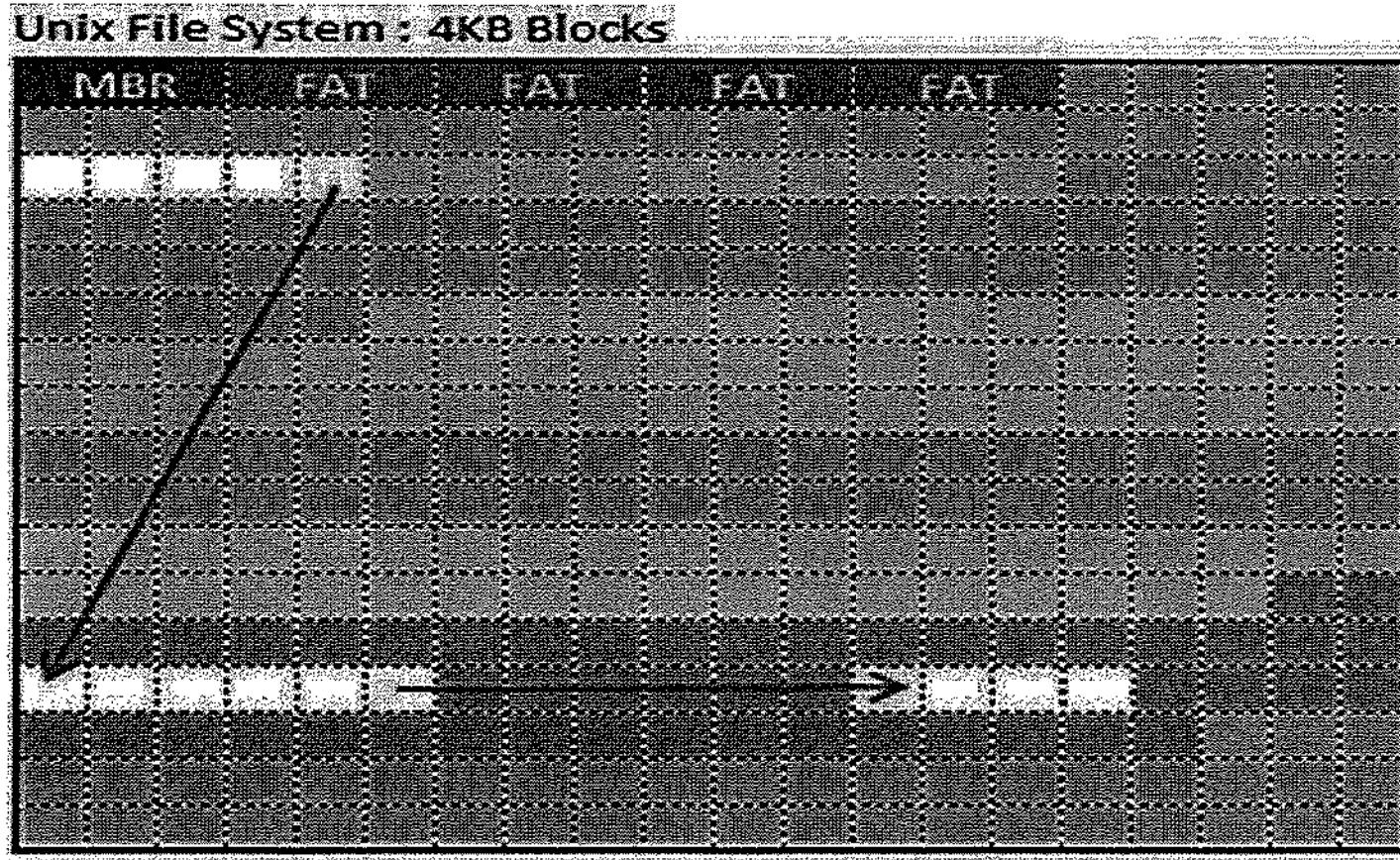
Unix File System : 4KB Blocks





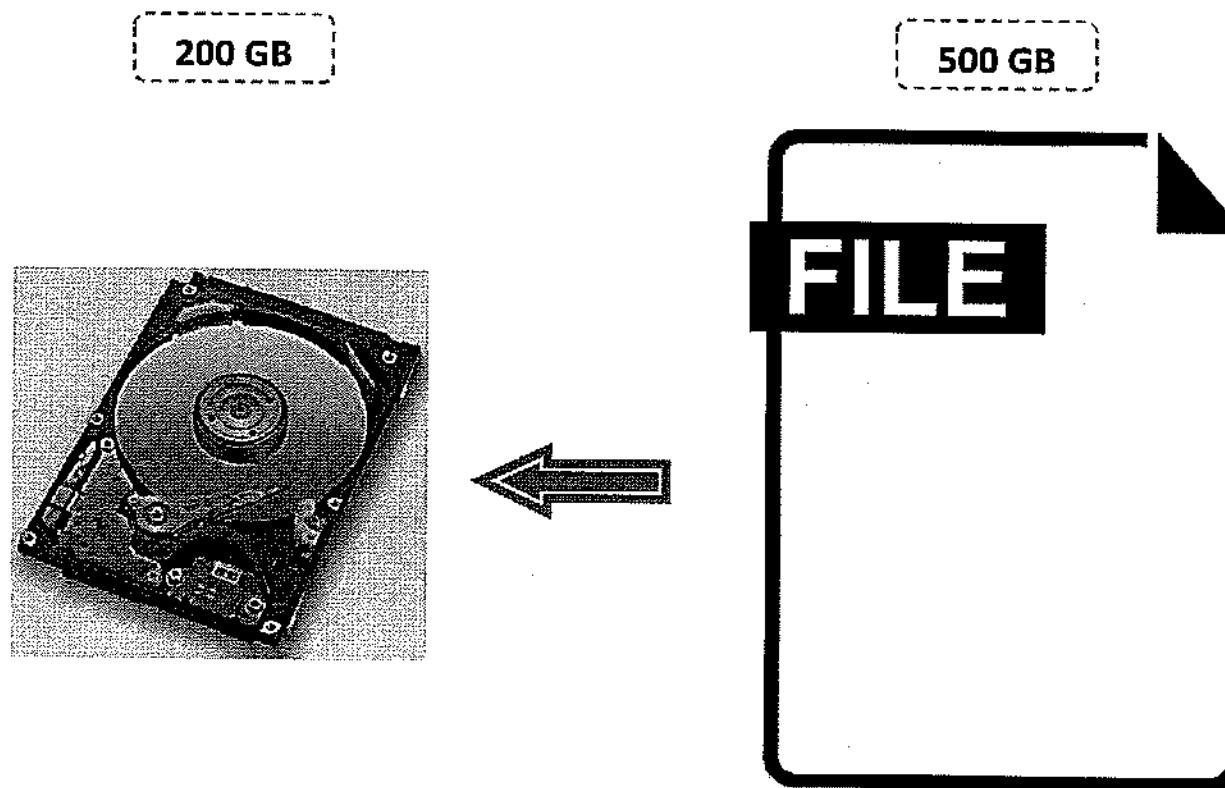
## What if my file is stored in non-continuous blocks?

- More number of seeks which causes delay in reading file blocks



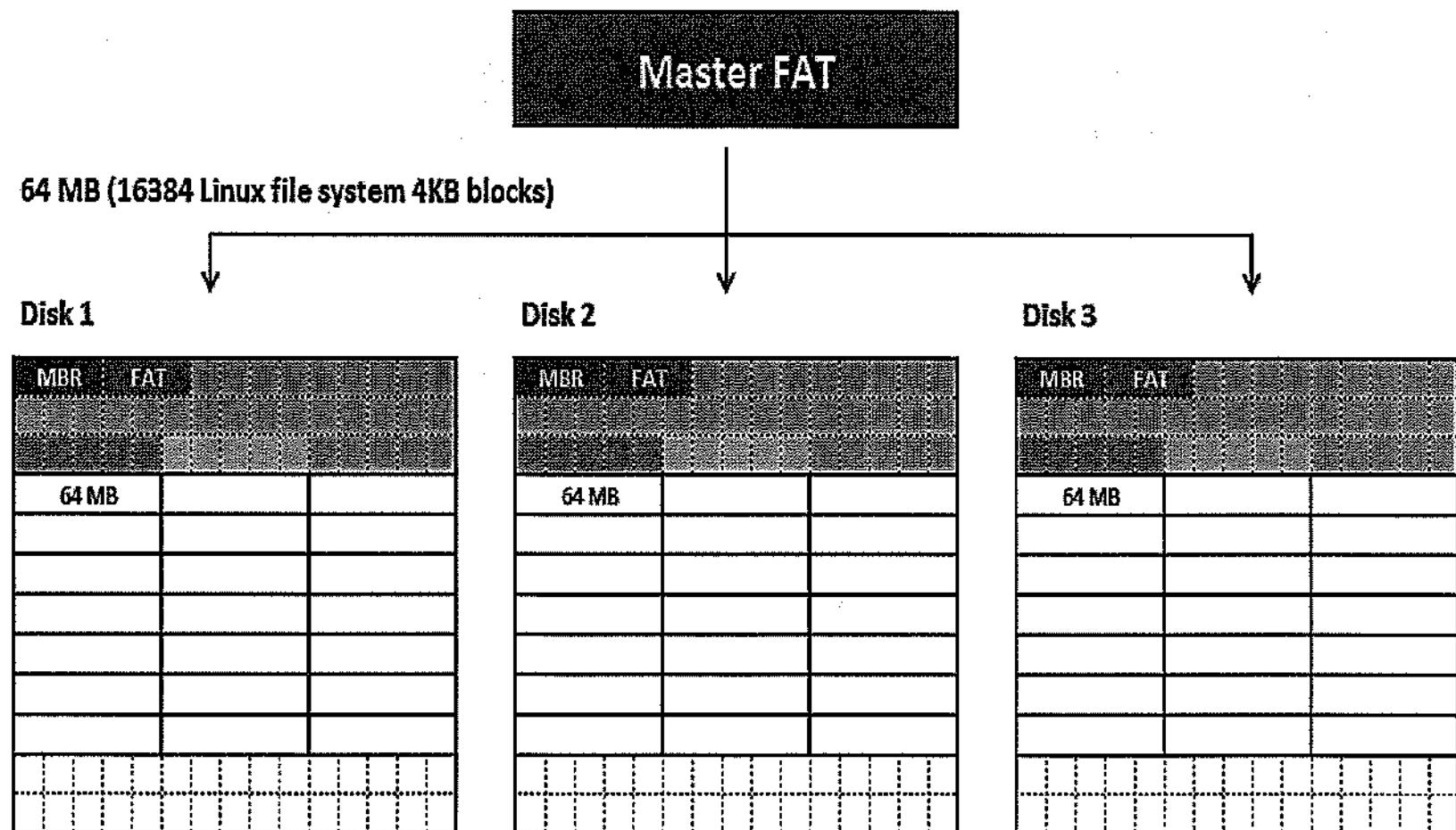


What if file size is larger than disk?



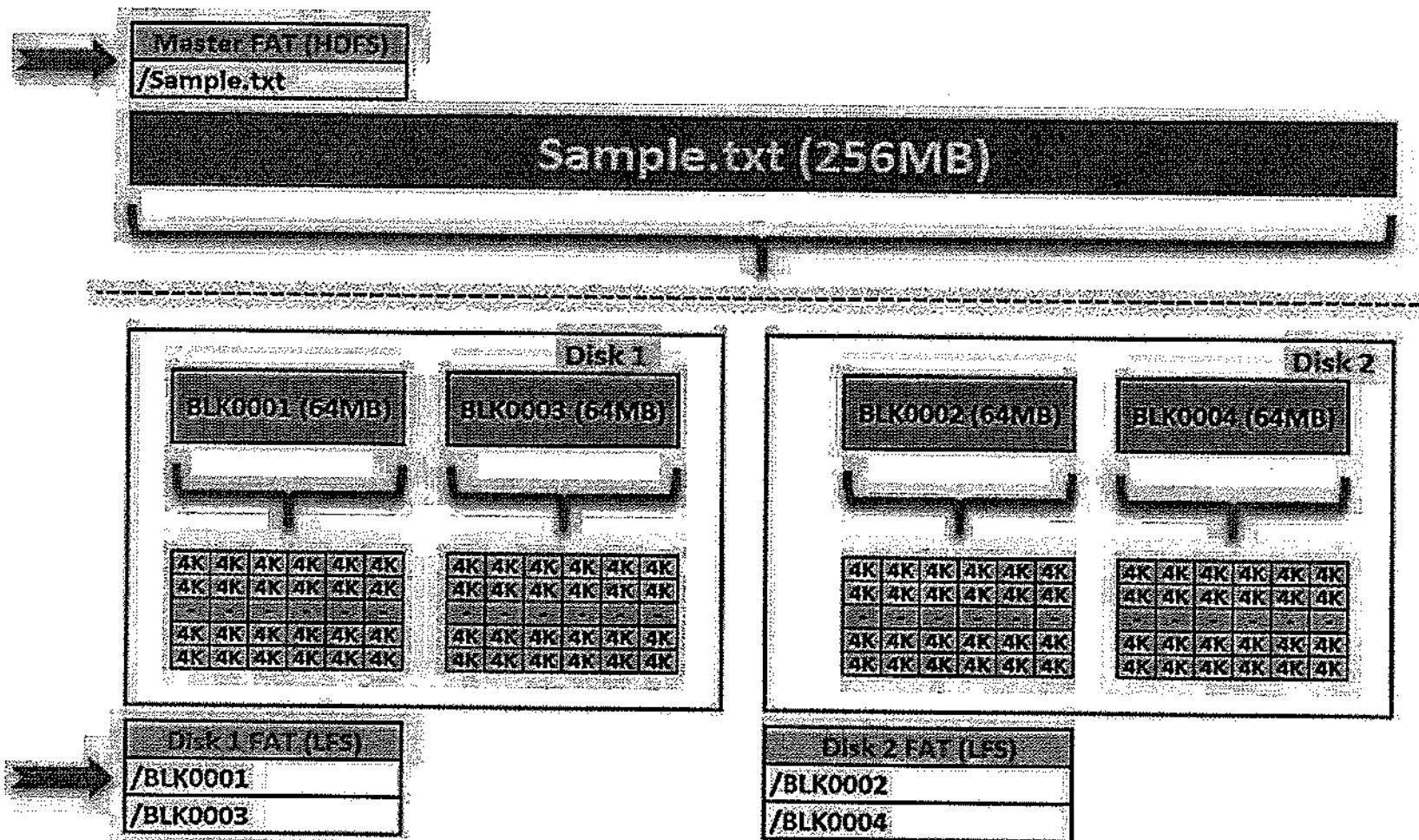


**What if we maintain Master FAT across multiple disks with larger block size?**





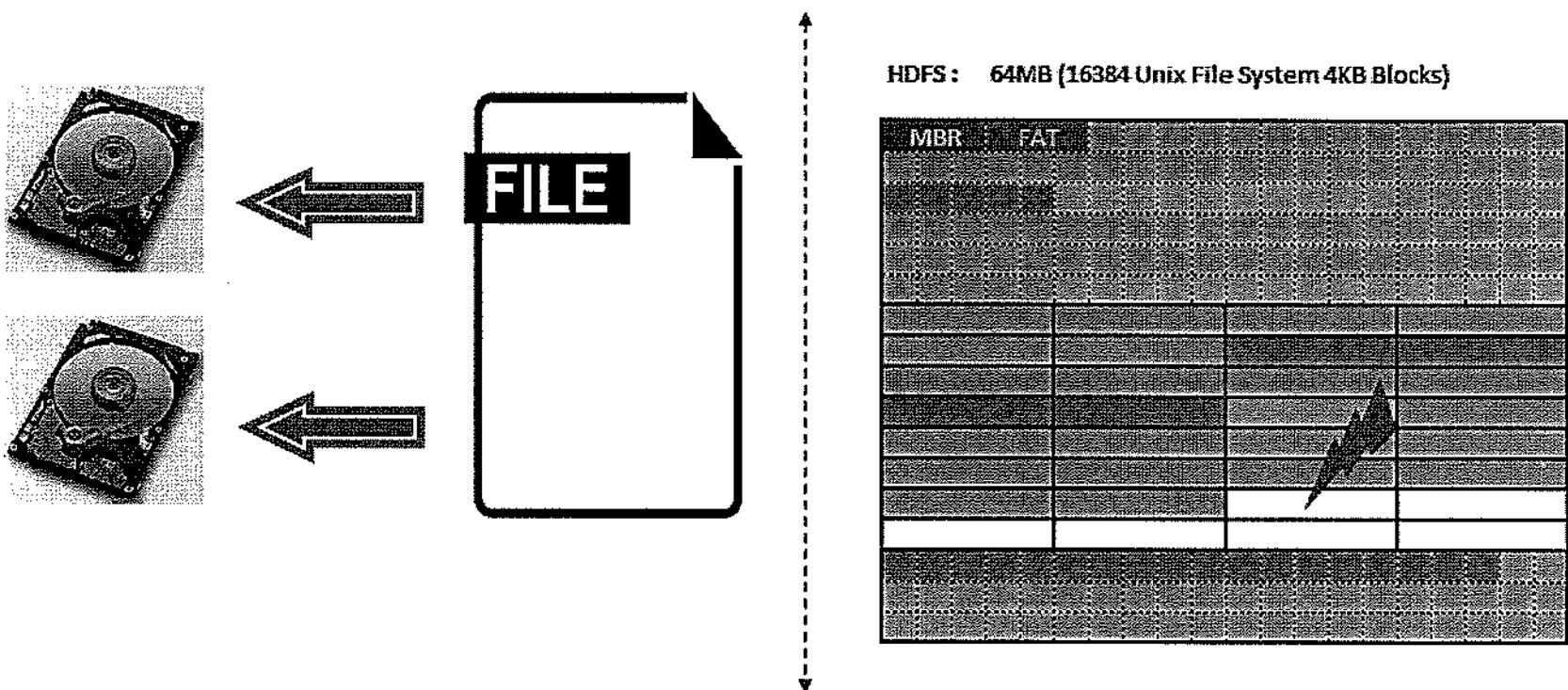
## Sample file storage on Master FAT





## Understanding Distributed File System

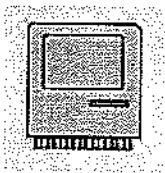
When a dataset/file outgrows the storage capacity of a single disk, it becomes necessary to partition it across multiple disks/servers. File systems that manage the storage *across a network of disks from single/different machines* are called **Distributed File Systems**.





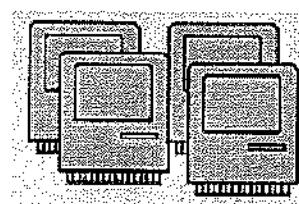
## What are the advantages with Distributed File System?

Let's say we have nearly **1 TB (1000 GB)** of data to write with a speed of **100 MB/S**



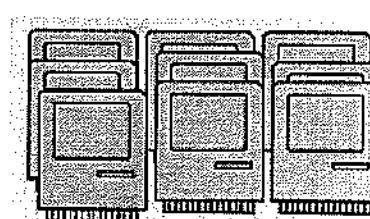
**1 Server**

- 1 Disk
- 1 TB per disk
- ***175 mins to write***



**10 Servers**

- 10 Disks
- 100 GB per disk
- ***17.5 mins to write***



**100 Servers**

- 100 Disks
- 10 GB per disk
- ***1.75 mins to write***





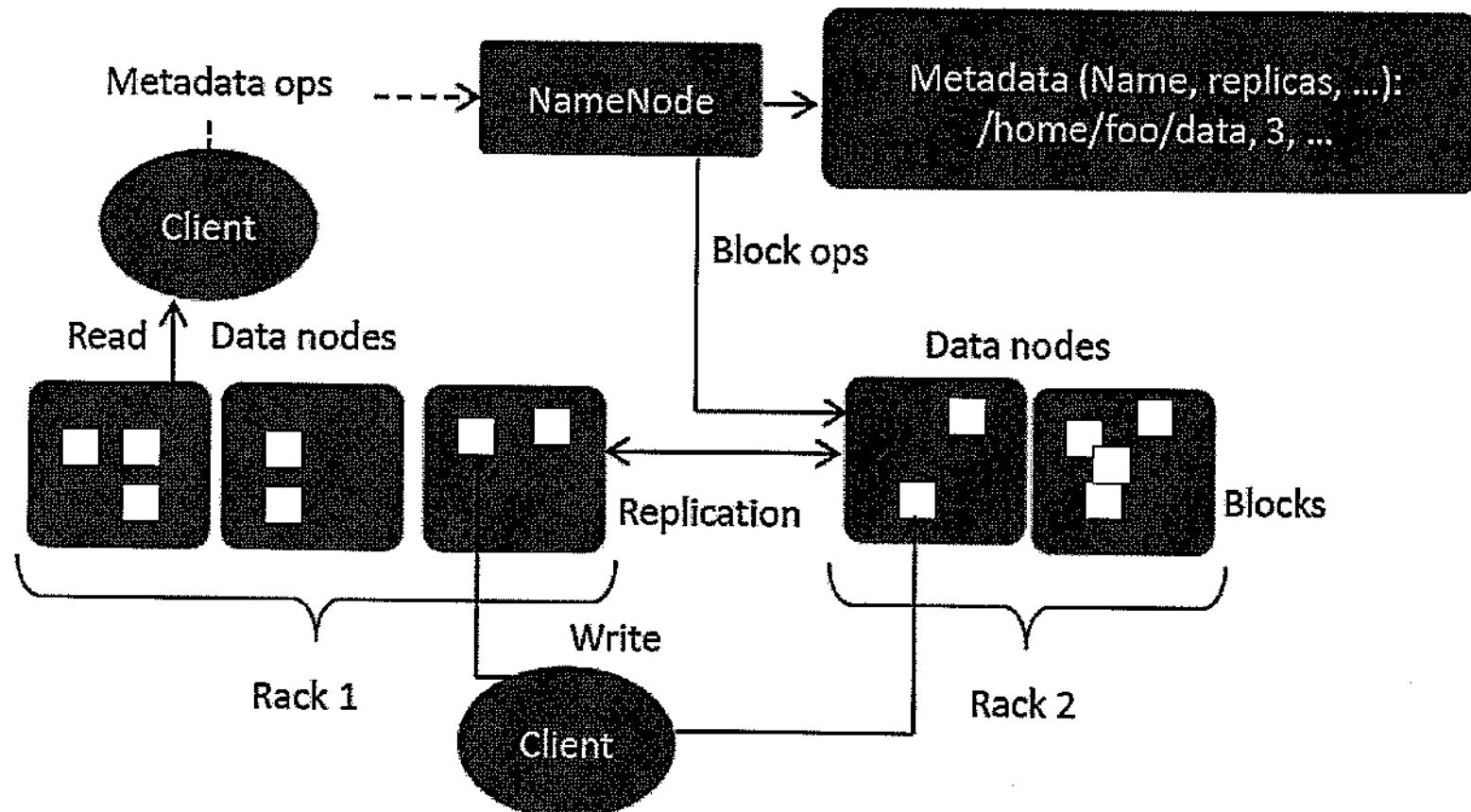
## Hadoop Components

- **HDFS – Hadoop Distributed File System**
  - *Storage Layer of Hadoop*
  - **Distributed file system that provides *high performance access to data with fault tolerance***
- **MapReduce – Programming Model**
  - *Processing Layer of Hadoop*
  - **Programming model with *Key & Value* pairs**





## Hadoop Architecture





## Storage Layer: HDFS

### NameNode:

- ***Master Node in Hadoop Cluster***
- **Responsible to store file system metadata**
  - **Information about file Name & Path**
  - **Information about file Ownership**
  - **Information about file Permissions**
  - **Information about file Timestamps**
  - **Information about file Size**
  - **Information about file Replication**
  - **Name of individual file Blocks**
  - **Locations of file Blocks**
- ***First point of contact for all HDFS operations***





## Storage Layer: HDFS (cont'd)

### DataNode:

- Slave Node in Hadoop Cluster
- Responsible to store actual data in the form of Blocks
- Each Block replica on a DataNode is represented by 2 files in the local file system

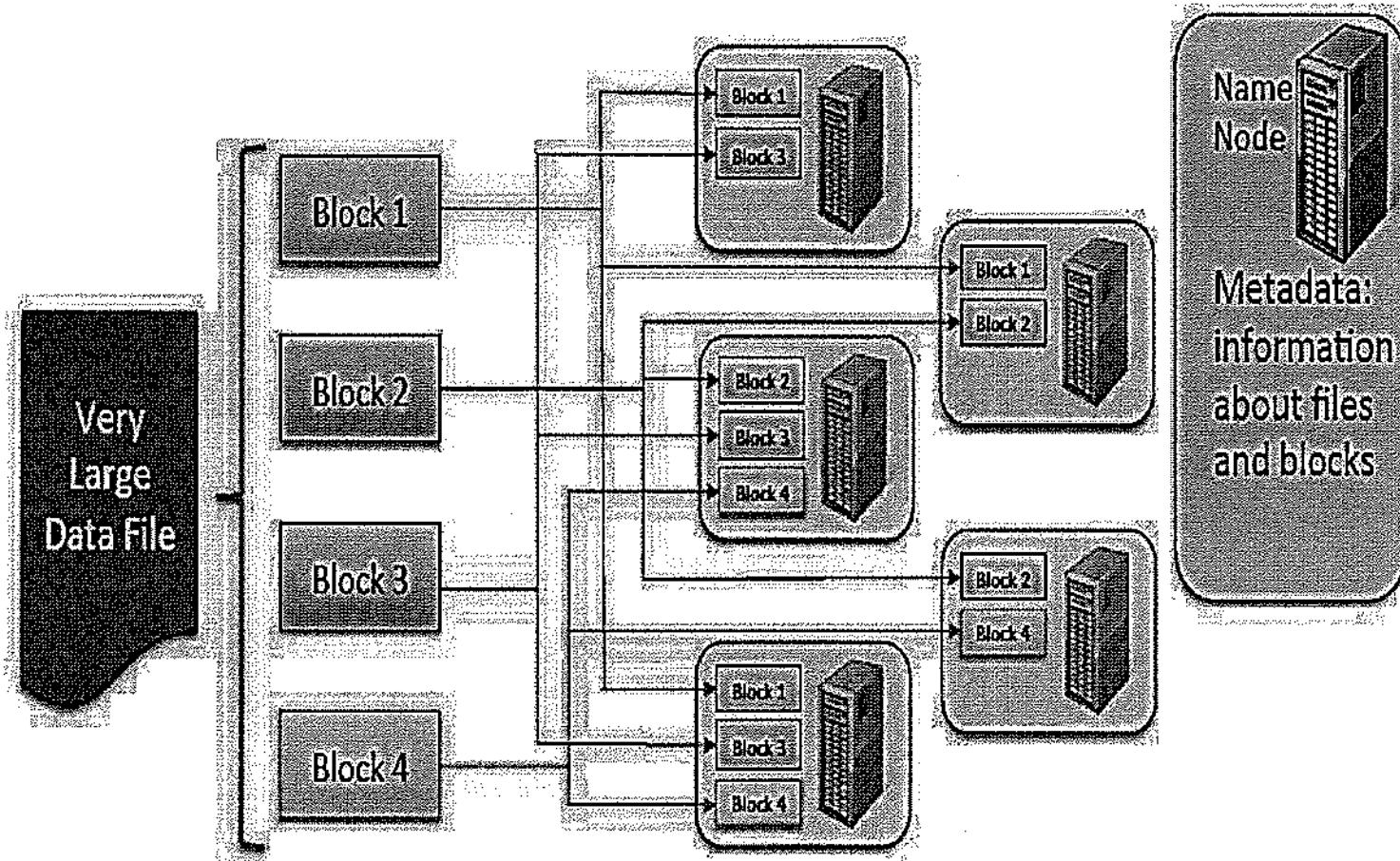
Ex:

```
-rw-r--r-- 1 hdpuser hdpadmin 1866 Jul 10 17:30 blk_1073741825
-rw-r--r-- 1 hdpuser hdpadmin    23 Jul 10 17:30 blk_1073741825_1001.meta
```

- The first file contains the *data* itself
- The second file contains block's *metadata* including checksum value, generation timestamps, etc.



## How file is stored on HDFS?





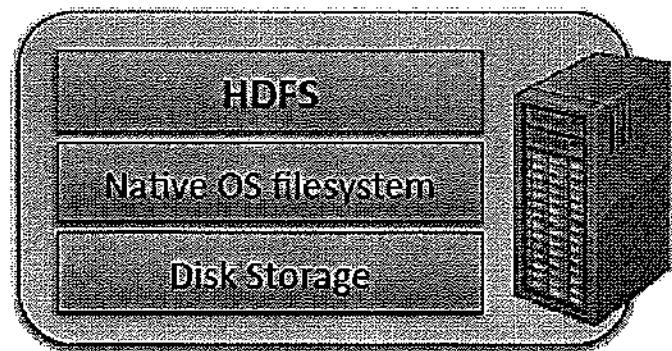
# Storage Layer (HDFS) and it's Internals . . .





## HDFS: The Hadoop Distributed File System

- HDFS is a file system written in Java
  - Based on Google's File System (GFS)
- Logical Layer on top of native file system
  - Such as *ext3, ext4 or xfs*
- Provides redundant storage for massive amount of data
  - Using industry-standard hardware
- At load time, data is distributed across all Nodes
  - Provides data processing efficiently

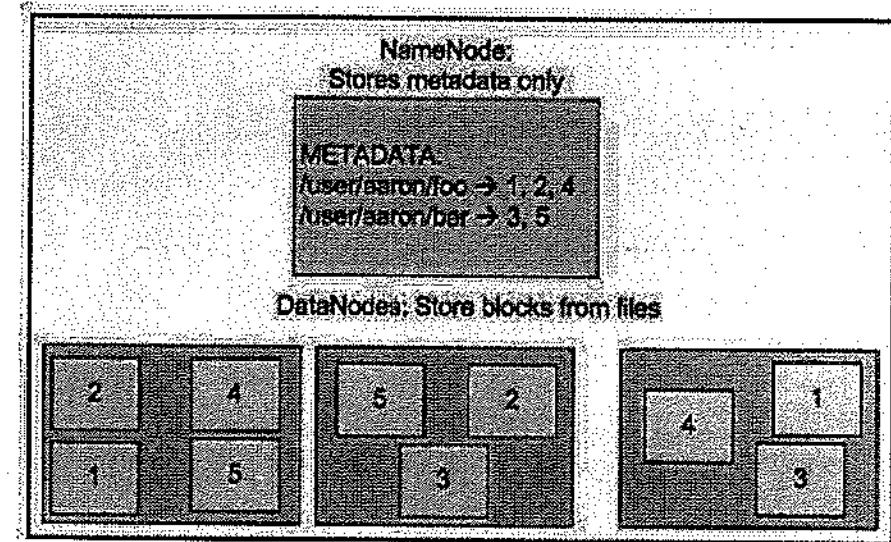




## HDFS Key Points

Default Block Size: **64 MB** (Hadoop 1.x)  
**128 MB** (Hadoop 2.x)  
**256 MB** (Hadoop 3.x)

- Every file is split into *Blocks*
- Every Block is *replicated 3 times by default*
- Block is *never replicated on the same Node*





## HDFS Features

- **High Performance**
- **Fault Tolerance**
- **Scalability**
- **Security**
- **Relatively simple centralized management**
  - **Master & Slave Architecture**
- **Optimized for distributed storage & processing**



## HDFS Design Assumptions

- Components may fail
- *Modest number of large files*
  - Hundreds of large files, not millions of small files
  - Each file is likely to be larger than block size in Hadoop
  - Typically, Multi-Gigabyte files
- Files are *write-once* and *read-many*
- Data can be appended to a file, but the file's existing contents cannot be changed
- Large streaming reads
  - Favor high sustained throughput over low latency disks



## Formatting the NameNode

- Can be formatted using *hadoop namenode -format* command
- In case of new Cluster, metadata files (*fsimage & edits*) will be created freshly
- In case of existing Cluster, old *fsimage* or *edits* will be truncated
- A unique *NameSpaceID* is assigned to Hadoop Distributed File System (HDFS)

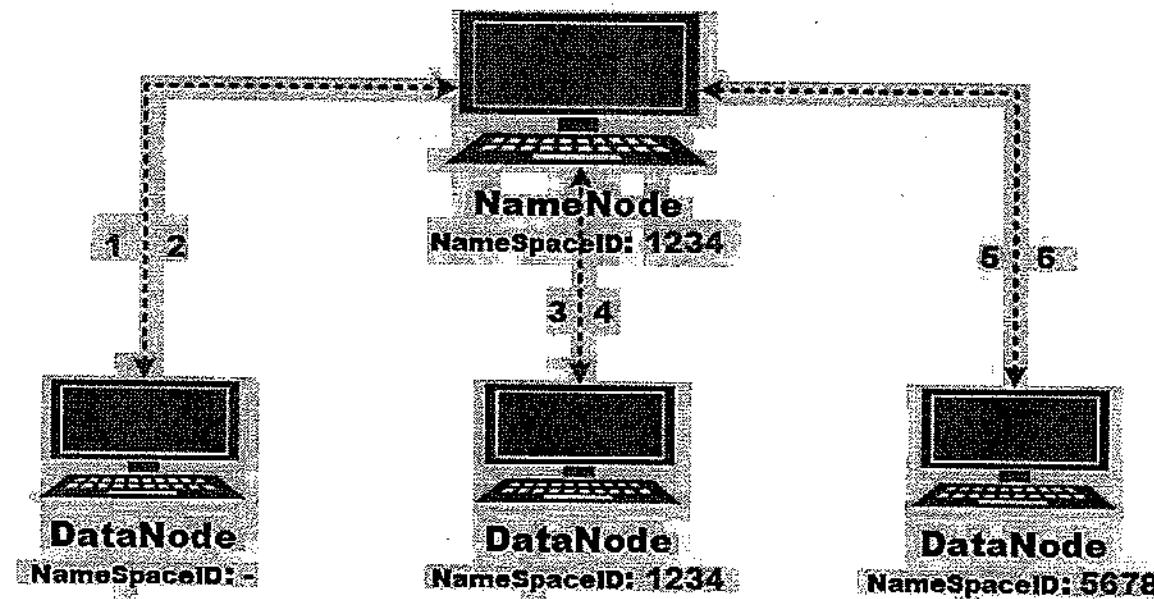
## What is the use of NameSpaceID?

A unique *NameSpaceID* is assigned to the Cluster when it is formatted. It does not change throughout the life of the Cluster. Purpose of this NameSpaceID is:

- Each DataNode will go for registration with the NameNode when the service is started
- The purpose of registration is to verify the NameSpaceID and the software version of Hadoop in the DataNode.
- DataNode service *automatically shuts down*, if there is a mismatch
- New DataNodes are allowed to register with NameNode and receive the NameSpaceID
- NameSpaceID is stored on all DataNodes upon successful registration with NameNode



## DataNode Registration with NameNode



1. Data Node Registration
2. No NameSpaceID (New Host). Registered with NameNode and gets the NameSpaceID
3. Data Node Registration
4. NameSpaceID matched (Already part of the Cluster), Registered with NameNode
5. Data Node Registration
6. Different NameSpaceID (Host is from different Cluster). Send *SHUTDOWN* signal to *DN service*



## Communication between NameNode and DataNode

- Each DataNode sends heartbeat to NameNode for every *3 seconds*
- DataNode sends Block Report & Free Space Info to NameNode over heartbeat periodically
- In response to DataNode's heartbeat, NameNode sends the required commands
- All Communication between NameNode and DataNode happens by using *RPC calls* through *TCP/IP port*

**NOTE:** NameNode *never* initiate the communication. In turn, it will send commands (like *on-demand Block Report, Block Invalidation, Block Replication, etc.*) in the same open channel of heartbeat.

## Communication between DataNodes

DataNodes can talk to each other by using *HTTP protocol*. This is mostly during block replication phase or map-reduce phase.





## Communication between Client and NameNode

If the Client is part of Hadoop cluster network, all Communication between Client and NameNode happens by using *RPC calls* through *TCP/IP port*.

If the Client is a Web Application, the communication will happen using Hadoop API calls (via *HTTP protocol*). Web Application Clients talk to NameNode whenever they wish to locate a file, or whenever they want to add/copy/move/delete a file on HDFS.

## Communication between Client and DataNode

All Communication between Client and DataNode happens by a *streaming protocol*.





## How metadata is maintained in Hadoop?

### *Fsimage:*

The entire file system metadata, including file properties and mapping of block(s) to file, is stored in a file called *fsimage*. NameNode maintains a copy of *fsimage* into its own memory to serve all Client requests effectively.

### *Edits:*

Rather than modifying the existing *fsimage* every time, NameNode instead records the new transactions (also called as *edits*) in the edit log for durability. Edit log transactions will be merged with *fsimage* on frequent intervals to bring *fsimage* content up-to date.

**NOTE:** *Fsimage* and *Edits* are *physical files* stored on *Local File System* of *NameNode* machine.

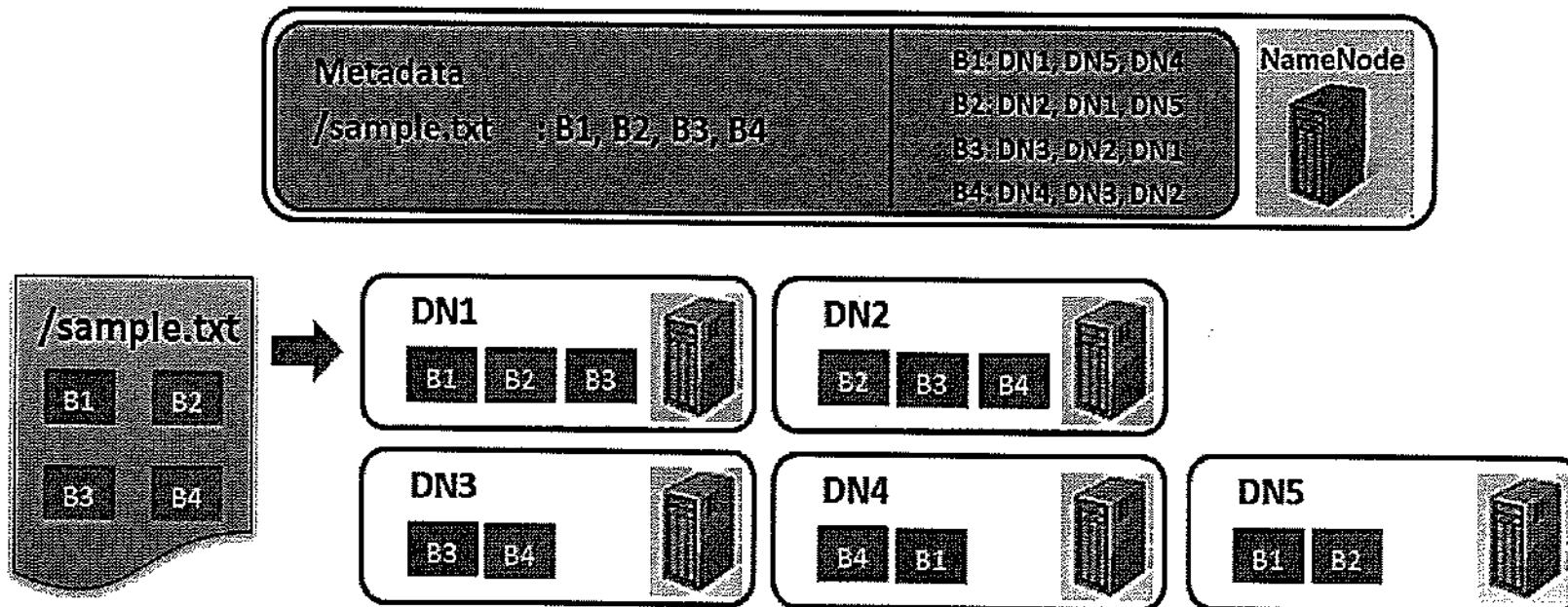




## Metadata in NameNode's Memory

List of Filenames, Owner, Permissions, Timestamps, Size, Replication, List of Blocks for each File, etc.

List of DataNode(s) for each Block (*Block Report*)





## What is Block Report?

- Whenever a DataNode service is started, it sends the list of blocks it has
- Block Report/Bitmap is a logical report which will be created in NameNode's memory and is *never saved to disk*

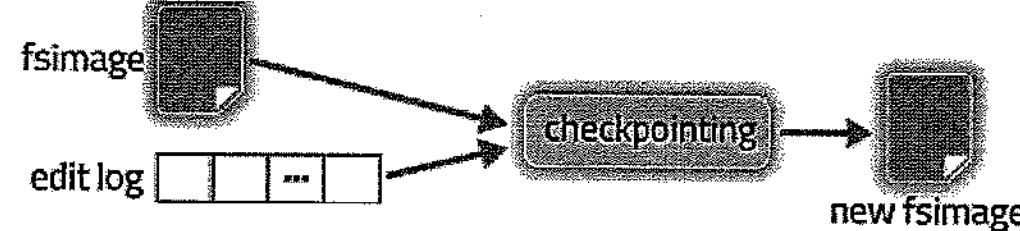
## When Block Report / Bitmap will be updated?

- Whenever NameNode / DataNode is started
- `dfs.blockreport.intervalMsec` (By default 1 hour in 1.x, 6 hours in 2.x)
- On-demand request from NameNode



## Checkpointing Mechanism

A process that takes *fsimage* & *edits* and combines them into a new *fsimage*.



## Who is responsible for Checkpoint?

- *Secondary NameNode* in Hadoop 1.X
- *Standby NameNode* in Hadoop 2.X / 3.X





## When Checkpoint will occur?

- Whenever NameNode is started
- `dfs.namenode.checkpoint.period`
  - (By default *60 mins*)
- `dfs.namenode.checkpoint.txns`
  - (By default *1000000 transactions*)
- `fs.checkpoint.size`
  - (By default *64MB*)
- Manual Checkpoint
  - (*hadoop dfsadmin -saveNamespace*)

## What happens if Checkpoint doesn't occur?

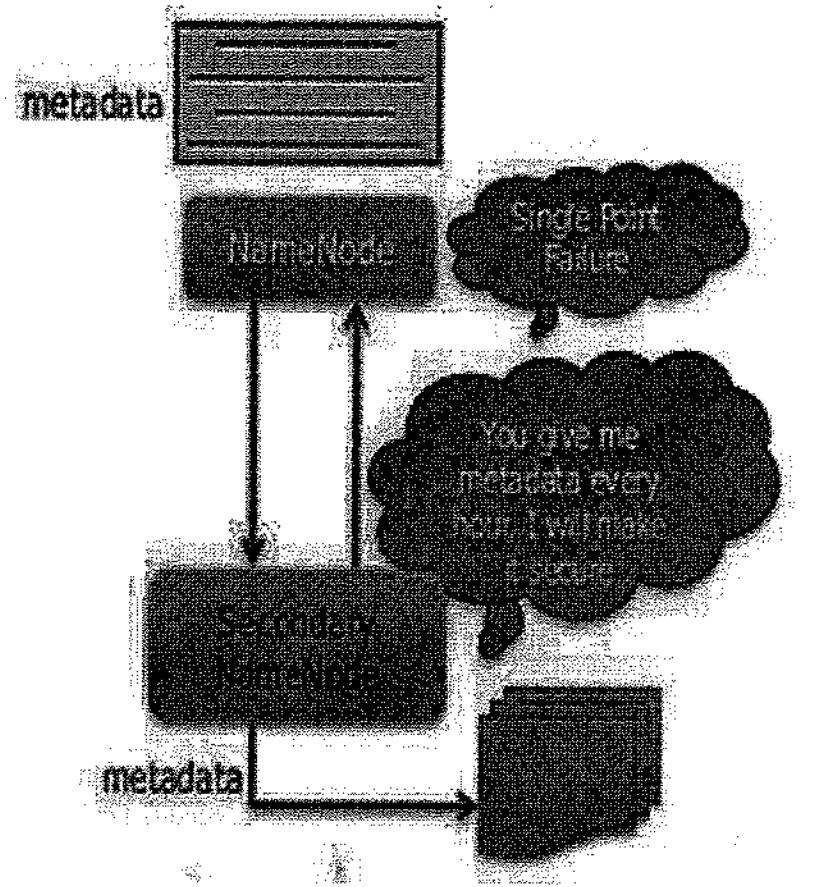
A couple of problems can arise if edit log grows bigger. In extreme cases,

- It can fill up all the available disk space on the Node
- It can delay NameNode startup as NameNode performs Checkpoint during startup



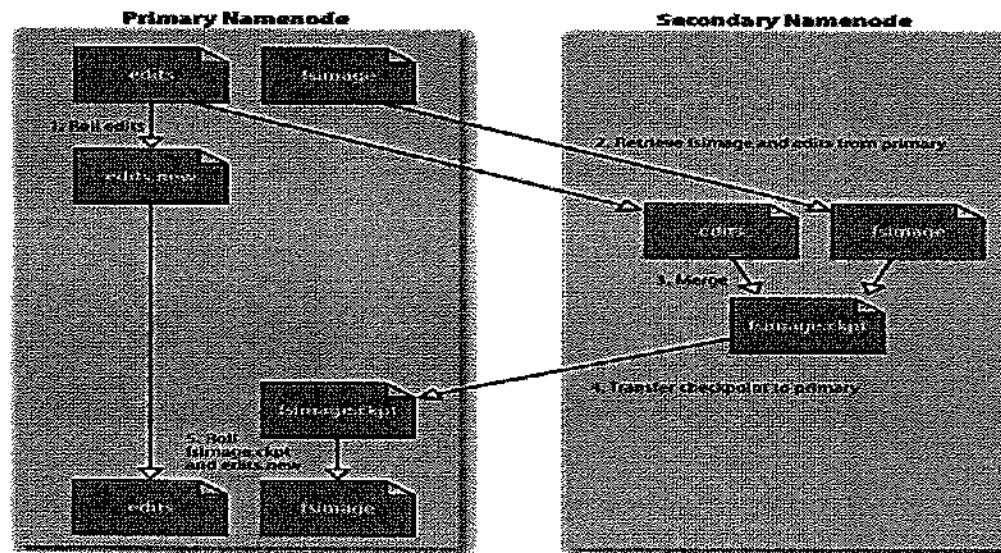
## Secondary NameNode

- Not a hot standby to the Primary NameNode
- Retrieve fsimage and edits from Primary NameNode and perform Checkpoint on regular intervals
- Maintains the latest copy of fsimage as backup of Primary NameNode's metadata



## How Checkpoint happens in the Cluster?

- Secondary NameNode creates *edits.new* file to log new transactions during Checkpoint
- Secondary NameNode Connects to Primary NameNode and pulls both *fsimage* & *edits*
- Merges both *fsimage* & *edits* into *new fsimage* and pushes it back to Primary NameNode
- This is done over *HTTP* channel



**NOTE: Secondary NameNode will *not* have latest copy of *fsimage* if checkpoint is performed by *NameNode* or *Manual Checkpoint (Scenarios 1 & 5 from previous slide)***





It's time for practicals on  
Storage Layer. .



## Installation Modes of Hadoop

### Standalone Mode

All Hadoop services run in a single Java Virtual Machine (JVM) on a single machine.



### Pseudo distributed Mode

Individual Hadoop services run in an individual JVM but on a single machine.

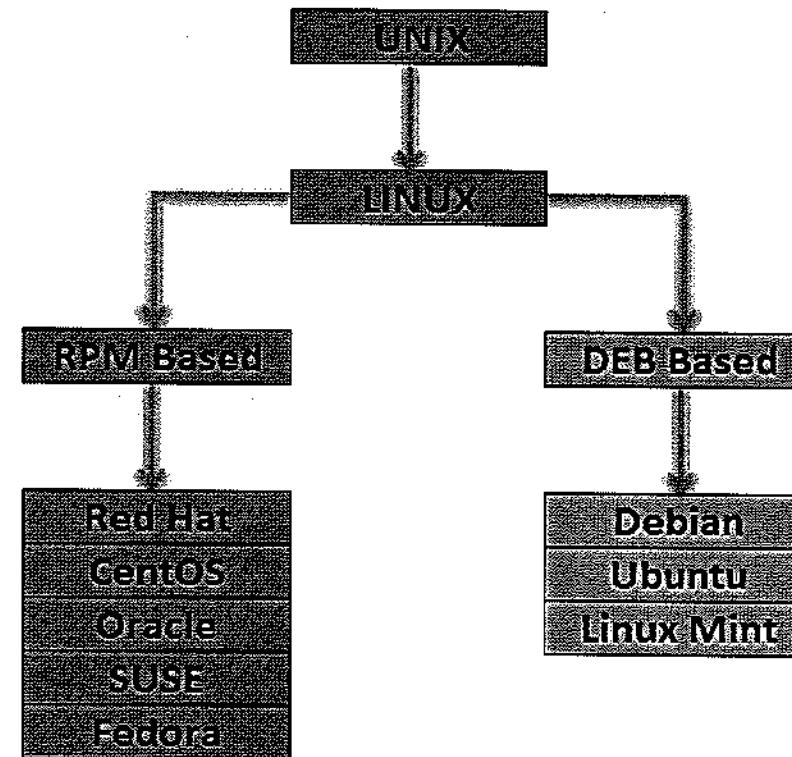
### Fully distributed Mode

Hadoop services run in different JVMs but in a cluster





## Linux OS Flavors





## Most commonly used Linux Commands in Hadoop Troubleshooting

- **File Commands:**

Command	Description
<code>ls</code>	<b>Directory Listing</b>
<code>cd dir</code>	<b>Change directory to <i>dir</i></b>
<code>pwd</code>	<b>Show current working directory</b>
<code>mkdir dir</code>	<b>Create a directory <i>dir</i></b>
<code>rm file</code>	<b>Delete <i>file</i></b>
<code>rm -r dir</code>	<b>Delete directory <i>dir</i></b>
<code>rm -f file</code>	<b>Delete <i>file</i></b>
<code>rm -rf dir</code>	<b>Force remove directory <i>dir</i></b>
<code>cp file1 file2</code>	<b>Copy <i>file1</i> to <i>file2</i></b>
<code>cp -r dir1 dir2</code>	<b>Copy <i>dir1</i> to <i>dir2</i>; Create <i>dir2</i> if it doesn't exist</b>
<code>scp</code>	<b>Copy from one server to another server</b>
<code>mv file1 file2</code>	<b>Rename or Move <i>file1</i> to <i>file2</i></b>
<code>ln -s file link</code>	<b>Create symbolic link to <i>file</i></b>
<code>cat file</code>	<b>Print <i>file</i> content</b>
<code>more file</code>	<b>Output the contents of <i>file</i> by page wise</b>





## Most commonly used Linux Commands in Hadoop Troubleshooting (Cont'd)

- *File Commands:*

Command	Description
<b>head file</b>	Output the first 10 lines of <i>file</i>
<b>tail file</b>	Output the last 10 lines of <i>file</i>
<b>tail -f file</b>	Output the contents of <i>file</i> as it grows, starting with the last 10 lines
<b>dos2unix</b>	Convert Windows file format to Unix file format
<b>md5sum file</b>	Print checksum value of a <i>file</i>

- *File Permissions:*

Command	Description
<b>chmod octal file</b>	Change the permissions of <i>file</i> to octal, which can be found separately for user, group, and world by adding: 4 – read (r), 2 – write (w), 1 – execute (x)
<b>chmod 777</b>	Read, Write, Execute for all
<b>chown file</b>	Change owner for the particular <i>file</i>
<b>chown -R dir</b>	Change owner for all files in the <i>dir</i>



## Most commonly used Linux Commands in Hadoop Troubleshooting (Cont'd)

- *System Information:*

Command	Description
<b>date</b>	Show the current date and time
<b>top</b>	Shows the top processes based on certain criteria like CPU or MEMORY
<b>uptime</b>	Show current uptime
<b>w</b>	Display who is online
<b>whoami</b>	Who you are logged in as
<b>ps -ef</b>	Provides the current processes along with detailed information
<b>uname -a</b>	Show kernel information
<b>cat /proc/cpuinfo</b>	CPU information
<b>cat /proc/meminfo</b>	Memory information
<b>cat /proc/loadavg</b>	Show system load average for the past 1, 5 and 15 minutes
<b>df -h</b>	Show disk usage
<b>du -sh</b>	Show directory space usage
<b>free</b>	Show memory and swap usage
<b>lsof</b>	List of open files
<b>which app</b>	Show which app will be run by default





## Most commonly used Linux Commands in Hadoop Troubleshooting (Cont'd)

- *Compression Related:*

Command	Description
<code>tar cf file.tar files</code>	Create a tar named <i>file.tar</i> containing <i>files</i>
<code>tar xf file.tar</code>	Extract the files from <i>file.tar</i>
<code>tar czf file.tar.gz files</code>	Create a tar with <i>Gzip</i> compression
<code>tar xzf file.tar.gz</code>	Extract a tar using <i>Gzip</i>
<code>tar cjf file.tar.bz2</code>	Create a tar with <i>Bzip2</i> compression
<code>tar xjf file.tar.bz2</code>	Extract a tar using <i>Bzip2</i>
<code>gzip file</code>	Compresses <i>file</i> and renames it to <i>file.gz</i>
<code>gzip -d file.gz</code>	Decompresses <i>file.gz</i> back to <i>file</i>

- *Network Related:*

Command	Description
<code>ping host</code>	Ping <i>host</i> and output results
<code>nslookup hostname/IP</code>	Check the <i>hostname/IP</i> of a machine
<code>netstat</code>	Display network connections (both incoming and outgoing)
<code>telnet</code>	To know whether the port on a particular server is listening or not
<code>wget file</code>	Download a <i>file</i>



## Most commonly used Linux Commands in Hadoop Troubleshooting (Cont'd)

- *Shortcuts:*

Command	Description
Ctrl+C	Halts the current command
Ctrl+Z	Stops the current command, resume with fg
Ctrl+D	Log out of current session, similar to exit
Ctrl+W	Erases one word in the current line
Ctrl+U	Erases the whole line
Ctrl+R	Type to bring up a recent command
exit	Log out of current session

- *Searching Related:*

Command	Description
grep pattern files	Search for pattern in files
find	Find file as per the given pattern
locate file	Find all instances of file
sed	Find and replace in file





## Preparing a Node for Hadoop Cluster

- Install OS with DHCP
- Disable SELinux, IPV6 & Others
- Disable Firewall or Open Hadoop Ports
- Install Hadoop Supporting RPMs (JAVA, openssh, python, httpd, etc.)
- Create Hadoop dedicated User & Group
- Configure Network with Static IP (DNS)

## Hadoop Manual Installation Steps

- Configure Password less SSH from Masters to all Slaves
- Enable Network Time Protocol Daemon (NTPD)
- Download and Extract Hadoop bundle
- Configure Environment Variables from OS level
- Configure Hadoop Configuration files
- Copy the modified Configuration files to all other Nodes in the Cluster
- Format NameNode (*One-time activity*)
- Start Hadoop Daemons



## Why Oracle JDK instead of Open JDK?

### Forking of JVM

- Light weight forking
- In Open JDK, if we set JVM size to 2GB and if it forks for child JVM, if no 2GB available, JVM will be crashed. This is not the case with Oracle JDK.

### JPS Command

- Available only from Java 1.6





## Hadoop Configuration Files

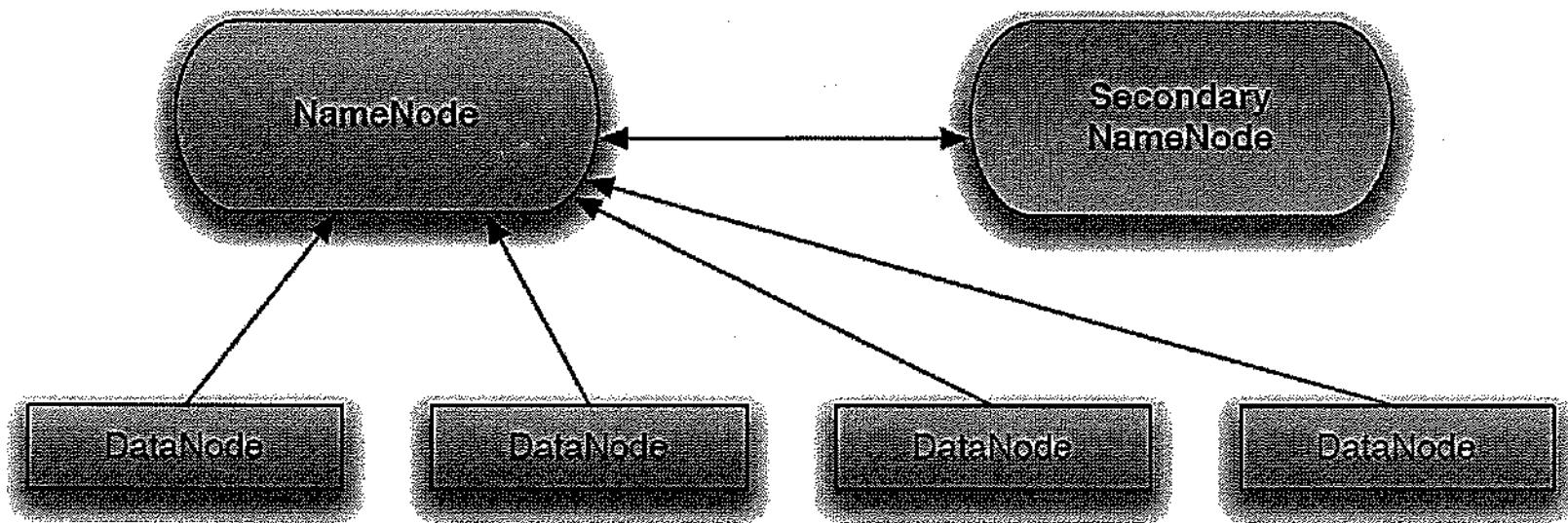
File Name	Description
hadoop-env.sh	Environment variables that are used in the scripts to run Hadoop
core-site.xml	Configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce
hdfs-site.xml	Configuration settings for HDFS daemons like NameNode, Secondary NameNode and DataNode
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
yarn-site.xml*	Configuration settings for YARN daemons: Resource Manager, the web app proxy server, and NodeManagers
yarn-env.sh*	Environment variables that are used in the scripts to run YARN (overrides variables set in hadoop-env.sh)
log4j.properties*	Properties for system log files, NameNode audit log, and the task log for the task JVM process
hadoop-policy.xml*	Configuration settings for access control lists when running Hadoop in secure mode

\* Files from Hadoop 2.X / 3.X





## Hadoop 1.X Architecture





## Limitations of Hadoop 1.X Storage Layer

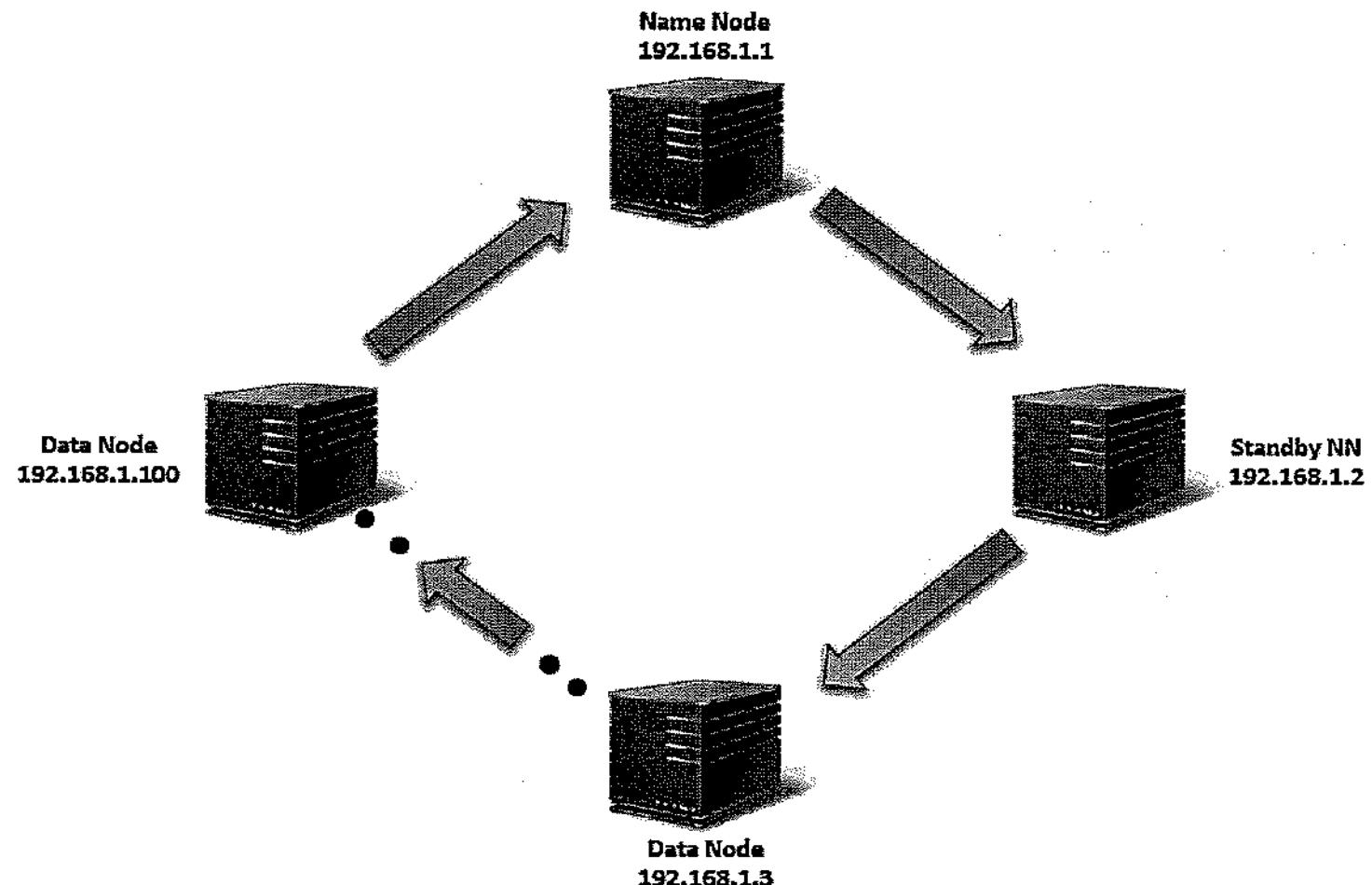
- NameNode is *Single Point of Failure*
- Secondary NameNode never acts as Primary NameNode in case of Primary NameNode failure
- Only one copy of Metadata is available for entire Cluster
- Chances of losing data if Checkpoint doesn't happen
- Metadata files need to be copied manually in case of Primary NameNode crash
- Limited to 4000 Nodes per Cluster

Solution: *Hadoop 2.X with NameNode High Availability*



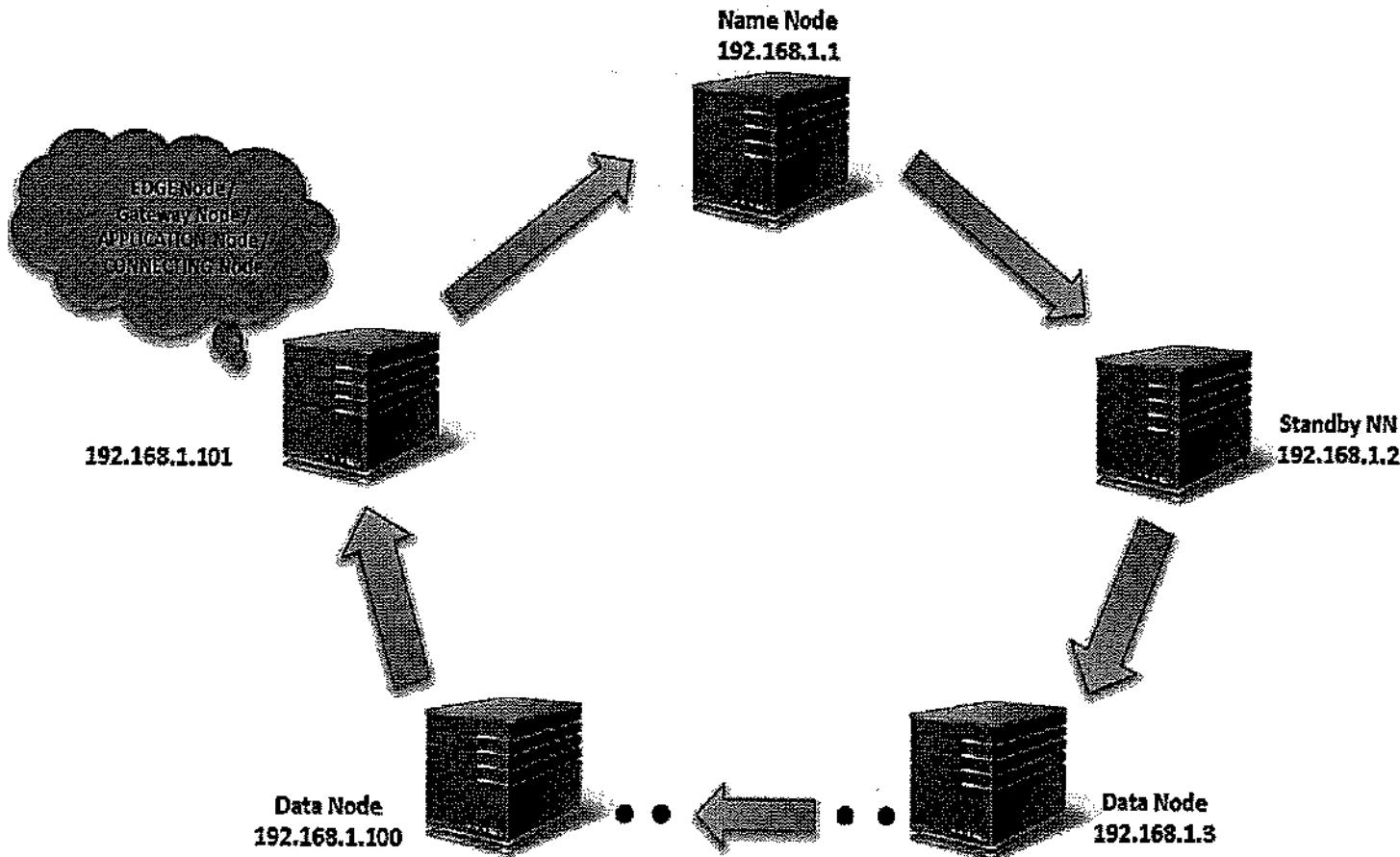


## What is Edge Node?





## What is Edge Node? (Cont'd)





## What is Edge Node? (Cont'd)

- Can be referred as Gateway / Application Node
- Edge Node is the interface between Hadoop Cluster and Users/Client Applications
- Users will be given access to Edge Node to execute their Hadoop commands/jobs
- Hadoop will be installed here but no Hadoop daemons will be running
- Often used as staging area for the data being transferred into Hadoop Cluster
- Hadoop Ecosystem Tools can be configured here

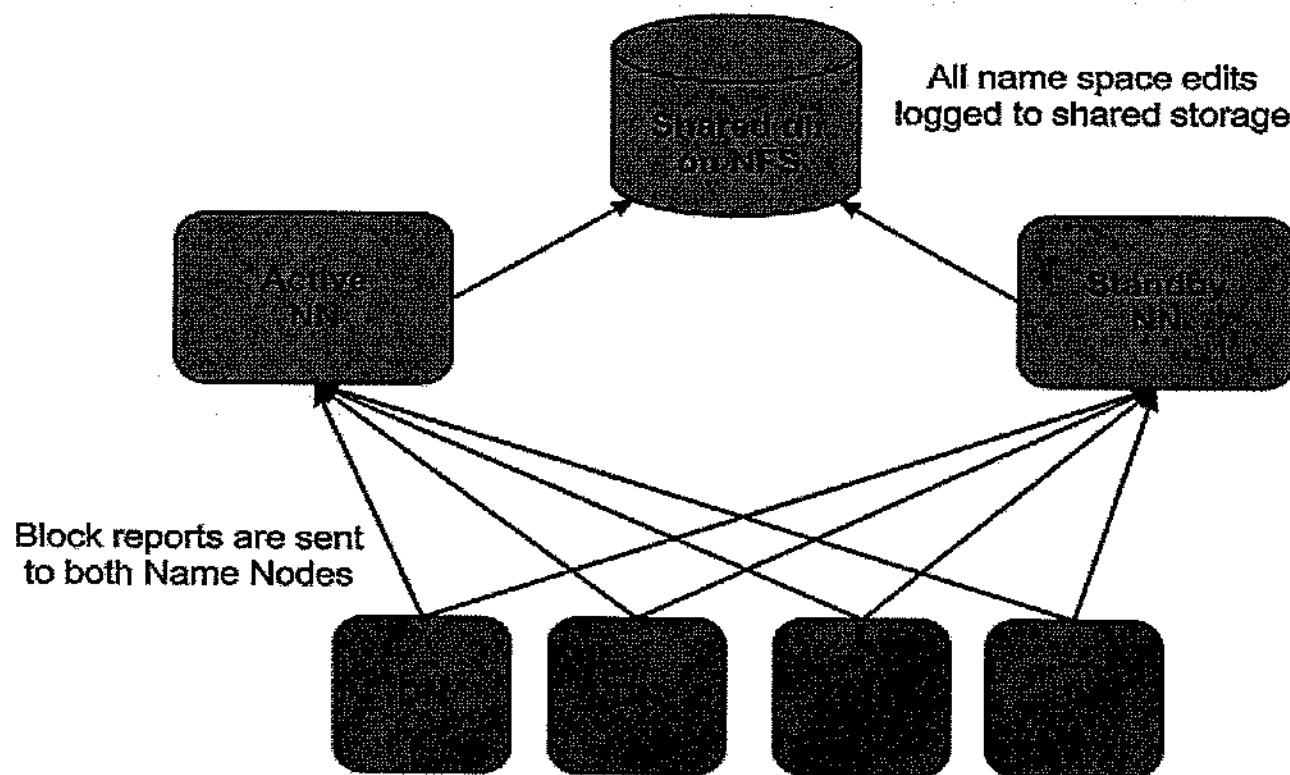
## Hadoop 2.X Setup differences in Storage Layer

- NameNode High Availability (*Active & Standby NameNodes*)
- No Secondary NameNode in Hadoop 2.X as the Standby NameNodes takes care of Checkpoint
- Common place to store edits for both NameNodes (*Using NFS Shared Filer / Journal Nodes*)





## Hadoop 2.X Architecture - NameNode HA (Using NFS Shared Filer)





## Limitations of NameNode HA using NFS Shared Filer

- Manual process of Active & Standby NameNode Election.

**Solution:** Additional mechanism (*ZooKeeper*) is required to automate this process.

- Slowness in writing Edits due to network overload
- No protection against hardware/network failure
- Only one common copy of Edits for both NameNodes.

**Solution:** Additional mechanism (*Journal Nodes*) is required to maintain multiple copies of Edits.





## ZooKeeper (ZK)

**ZooKeeper (ZK) is a highly-available, highly-reliable and fault-tolerant coordination service for distributed applications.**

***Ensemble*** : A group of ZooKeeper Nodes worked as a single unit rather than individual

***Quorum*** : No. of Nodes that should be up and running in the ensemble

**Number of Node failures that can be tolerated in the ensemble will be  $ROUND(N/2, 0) - 1$ , where "N" is the number of total Nodes in ensemble.**

## What is ZNode?

**ZooKeeper maintains an active connection with all its Clients using a heartbeat mechanism.**

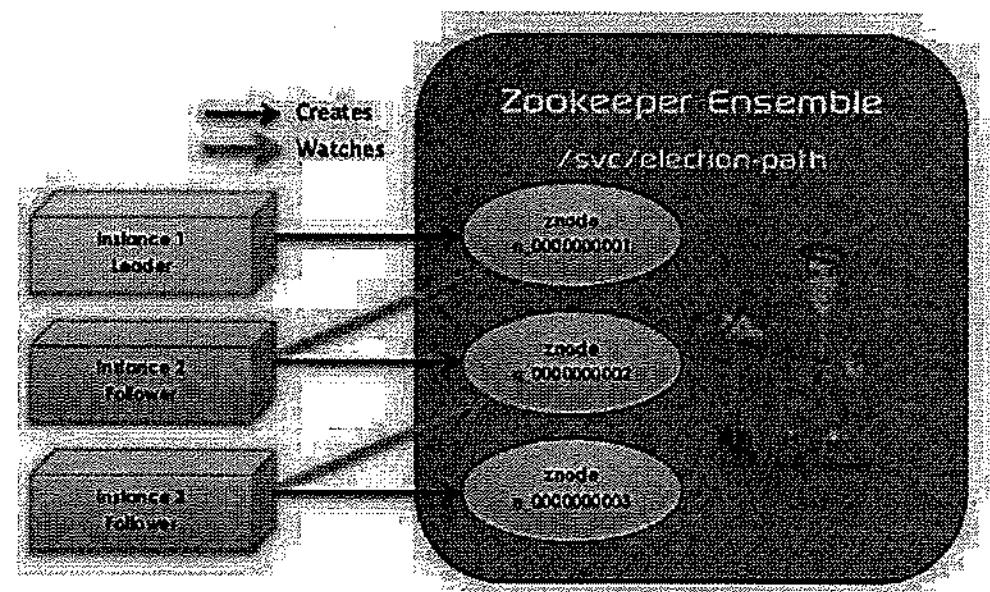
**Furthermore, ZooKeeper keeps a session for each active Client that is connected to it and assigns a key which is called ZNode.**

## What are the types of ZNodes?

- **Persistent** – alive even after its Client session is disconnected
- **Ephemeral** – alive only for the lifetime of its Client session
- **Sequential** – Sequential ZNodes can be either Persistent or Ephemeral

## ZooKeeper Leader Election

Participants of the *ZooKeeper Leader Election* process create an *ephemeral-sequential* Node in the election path. The Node with the smallest sequence number is the “Leader”. Each “Follower” Node listens to the Node with next lower sequence number. If Leader goes down, the follower corresponding to the Leader will become the new Leader.





## Why odd number of Nodes in ZK Ensemble?

ZK Ensemble of 5 Nodes:  $\text{Round } (5/2, 0) - 1 = 2$

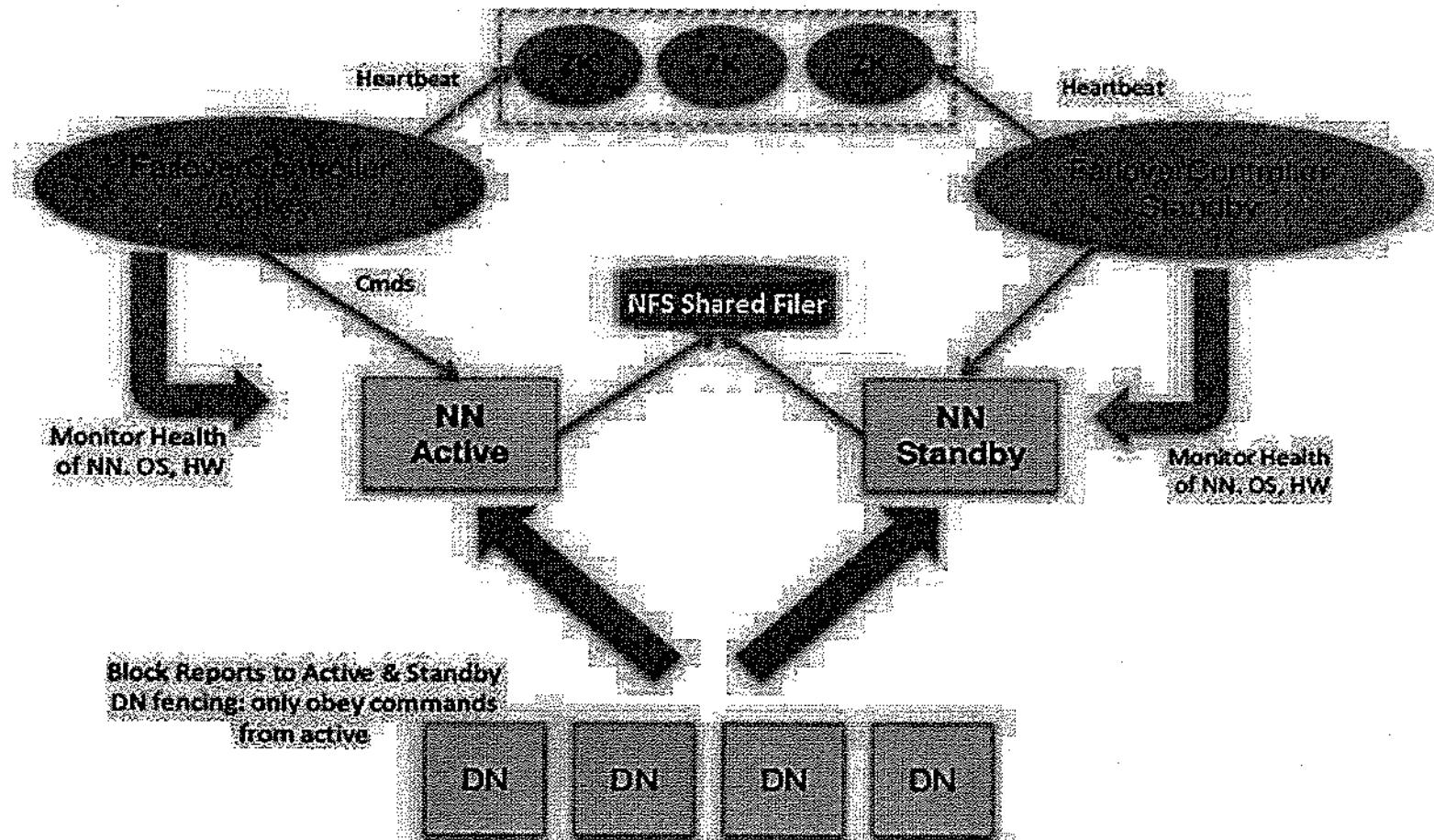
ZK Ensemble of 6 Nodes:  $\text{Round } (6/2, 0) - 1 = 2$

The extra Node doesn't add any benefit for the Cluster. So, replicating to that one extra Node is just a performance overhead.

## Performance Tuning in ZooKeeper

- Increasing number of ZooKeeper connections (*maxClientCnxns*) in *zoo.cfg* is used by the ZooKeeper server to limit incoming connections to the ZooKeeper from a single host. By default, this value is 60.
- Better to have 5 ZooKeepers in Ensemble instead of 3 for more High Availability during patching & upgrades. Again, the more ZK nodes you have, the slower the ZK becomes for writes.
- Enable auto-purge option for the ZooKeeper snapshots data to not to increase space on disk.
- JVM size should be chosen with care and do not put ZooKeeper in a situation that can cause a swap.

## Hadoop 2.X Architecture - NameNode HA (Using ZooKeeper & NFS Shared Filer)



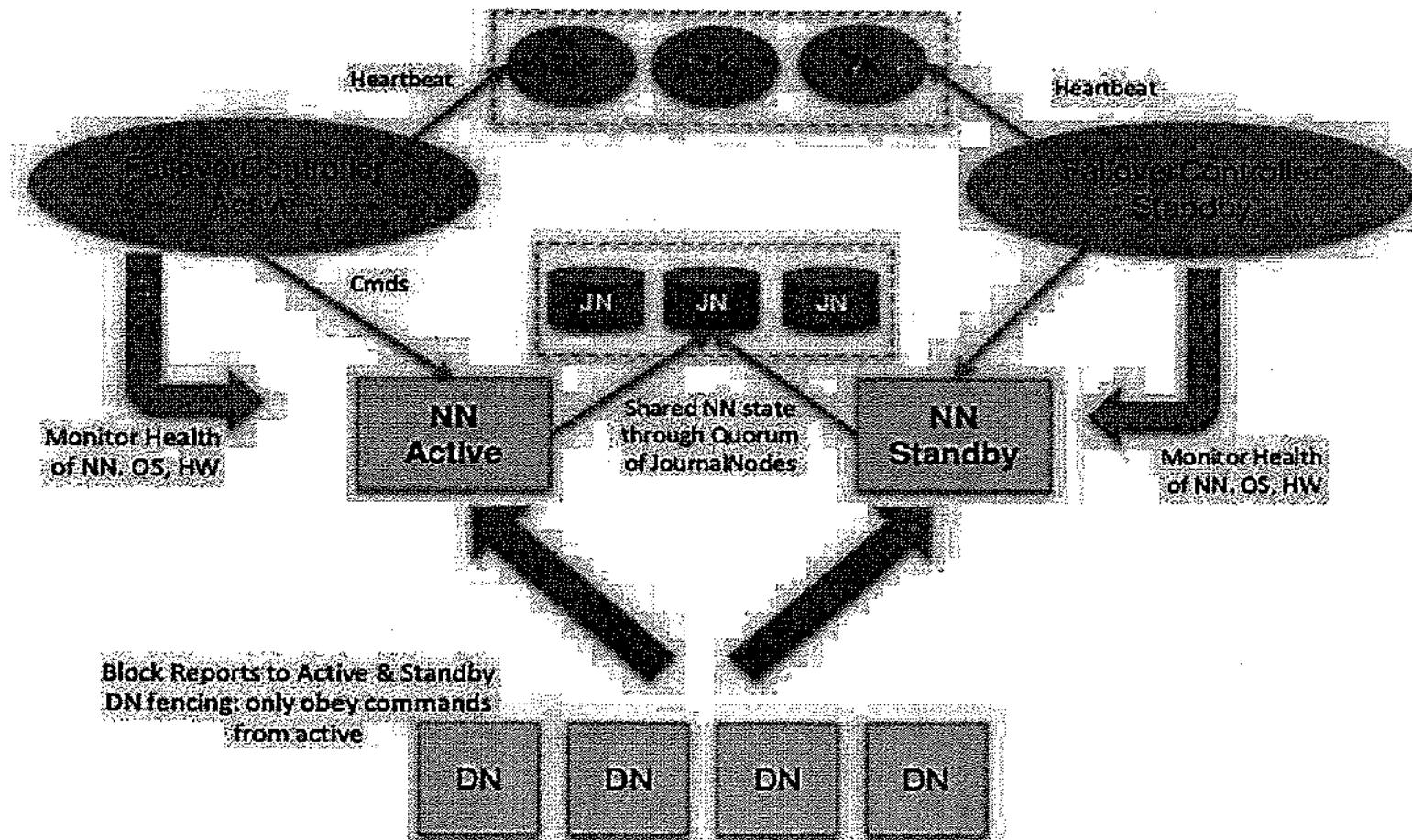


## Journal Nodes (JN)

In order to keep Standby NameNode's state synchronized with Active NameNode, both NameNodes communicate with a group of separate daemons called *Journal Nodes* (JNs). When a new transaction (also called *Edit*) is performed by Active NameNode, it durably logs it into a majority of these Journal Nodes. The Standby NameNode is capable of reading these transactions from the Journal Nodes. During Checkpoint time, Standby NameNode applies the latest transactions to its own Fsimage. In the event of a failover, the Standby NameNode will ensure that it has read all the transactions from the Journal Nodes before promoting itself to the Active state. This ensures that the metadata state is fully synchronized before a failover occurs.



## Hadoop 2.X Architecture - NameNode HA (Using Journal Nodes & ZooKeeper)





## How to change edits from NFS Shared Filer to Journal Nodes?

- Perform manual Checkpoint
- Add Journal Node properties to *core-site.xml* file
- Add Journal Edits location properties in *hdfs-site.xml* file
- Start Journal Nodes and stop Standby NameNode
- Initialize the Shared Edits (*hdfs namenode -initializeSharedEdits*)
- Copy the NFS Shared Edits from NFS to Journal Node location in all Journal Nodes
- Start Standby NameNode
- Failover the NameNodes
- Stop & Start the current Standby NameNode
- Make sure both the NameNodes are pointed to Journal Nodes

**NOTE:** Since changing of edits from NFS Shared Filer to Journal Nodes is a *migration* process, it is always preferable to have a scheduled downtime for your Cluster while performing this activity.



## Advantages of NameNode HA with ZooKeeper & Journal Nodes

- Automatic failover to Active and Standby by using ZooKeeper
- High Availability against hardware / network failure
- Multiple copies of Edits for both the NameNodes

### Failover

The transition from NameNode's active state to the standby is managed by a new entity in the system called the *Failover Controller*. Each NameNode runs with a lightweight ZooKeeper Failover Controller (*ZKFailoverController / ZKFC*), whose job is to monitor its NameNode for failures (using a simple *heartbeat* mechanism) and triggers a failover.

**Graceful failover:** Failover may also be initiated manually by an administrator (Example: In case of routine maintenance).



## Fencing

**Fencing:** In case of an ungraceful failover, however, it is impossible to be sure that the failed NameNode has stopped running (*Example:* A slow network or a network partition can trigger a failover transition). In this case, the previously active NameNode will be still in Active state and try to write the Edits in Journal Nodes. With the help of Fencing mechanism, HA implementation will ensure that the previously active NameNode is prevented from doing any damage and causing corruption.

**SSH Fencing** is a method to kill the previously active NameNode's state using SSH Private Key.

**SHELL Fencing** is a method to kill the previously active NameNode's state using script.

## What if we make both the NameNodes ACTIVE?

If both the NameNodes are active, they think that they are in control and both try to write to the same Edit Log. There are chances of corruption of metadata in this case. This is like "*split-brain*" syndrome issue. Fencing in ZKFC will take care of this issue by *killing* one of the NameNode's Active state.



## HDFS Federation

HDFS federation, introduced in the 2.x release series, allows a Cluster to scale by adding multiple NameNodes, each of which manages a portion of the file-system namespace. For example, one NameNode might manage namespace for all the files stored under */Project1*, and second NameNode might manage namespace for all the files stored under */Project2*.

Under federation, each NameNode manages a *namespace volume*, which is made up of the Metadata for the namespace, and a *Block Pool* containing all the blocks for the files in the Namespace. Namespace volumes are independent of each other, which mean NameNodes do not communicate with one another, and furthermore the failure of one NameNode does not affect the availability of the namespaces managed by other NameNodes. Block Pool storage is *not partitioned*, so DataNodes register with each NameNode in the Cluster and store blocks from multiple Block Pools.





## Hadoop 3.X Setup differences in Storage Layer

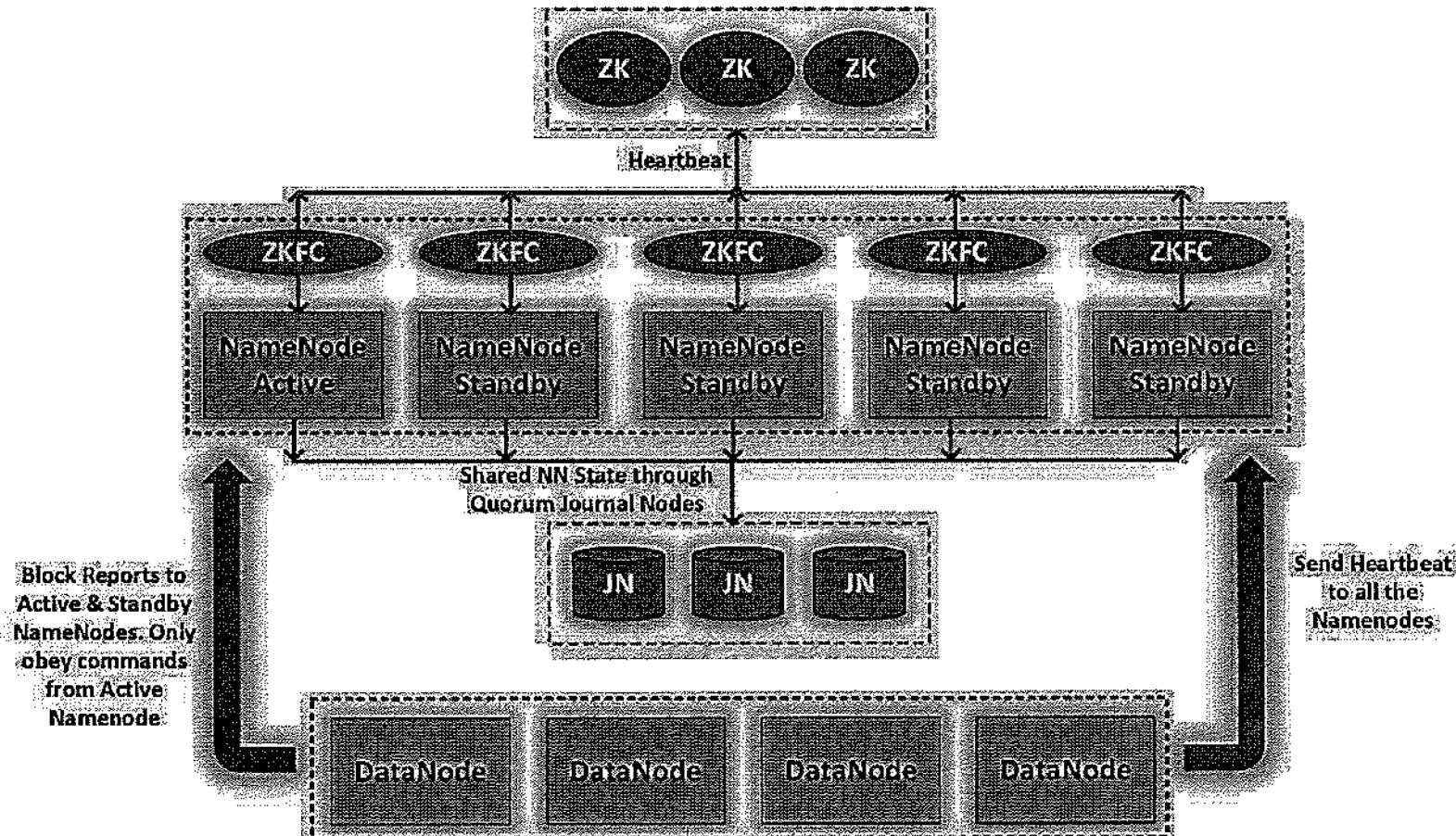
Majorly,

- Minimum required Java version is **Java 1.8**
- Default Block size is **256MB**
- Support for more than 2 NameNodes (**1 Active & Multiple Standby NameNodes**)
- Support for Erasure Encoding in HDFS (requires storage overhead up to **50%** instead of **200%**)
- Intra-DataNode Balancer
- Can be scaled for more than **10,000 nodes per cluster**
- Difference in default port numbers (Ex: NameNode port is **9870** instead of **50070**)
- ***hdfs --daemon <stop/start> <service name>*** (No *hadoop-daemon.sh*)
- Supports Microsoft Windows & Microsoft Azure Data Lake filesystem





## Hadoop 3.X Architecture





# Some interesting points about HDFS and it's Internals ..





## NameNode Startup Process

During startup, NameNode must complete certain actions before it serves Client requests:

- Read file system metadata from *fsimage*
- Read logged transactions from *edits*
- Perform checkpoint (creating a new *fsimage* which consists prior *fsimage* and recent edits)
- Remain in safe mode until a sufficient number of blocks have been reported by DataNode(s)

In some situations, these actions can take a long time to complete. For example:

- Large edit logs due to a long-term outage of Secondary/Standby NameNode
- A degraded disk can slow down performance
- When you have multiple checkpoint locations
- When you start all DataNode services together in a large Cluster

**Solution: Rolling Start/Restart of DataNode services**





## Hadoop Safemode

Safemode is the mode in which Cluster is in read-only mode. This is when the Checkpoint is done and it builds the "*Block Report / Bitmap*". We cannot perform any write operations when Cluster is in Safemode.

### Commands:

`hdfs dfsadmin -safemode get`

`hdfs dfsadmin -safemode enter`

`hdfs dfsadmin -safemode leave`

`hdfs dfsadmin -safemode wait` (wait for some more time till bitmap ready. Default 5mins)

`hdfs dfsadmin -saveNamespace` (to perform *checkpoint*)

`hadoop secondarynamenode -checkpoint`

`hadoop secondarynamenode -checkpoint force`

`hdfs dfsadmin -metasave filename.txt`



## NameNode Heap Memory requirements to store metadata

- When NameNode is running, a copy of its metadata will be held in RAM for faster response
- Default Java Heap Size of the NameNode is 1GB
- At least 1GB recommended for every *million* HDFS blocks
  - Ex: 1GB is required for 120TB of data with a block size of 128MB
- Items stored by the NameNode:
  - Fsimage : Filename, Owner, Permissions, Timestamps, Size, Replication, Block numbers etc.
  - Block Report : Location information for each block

### Metadata Storage Requirements for a file with 2 blocks

Object	Space Required	
File Name	150 Bytes	150 Bytes
Block 1	1 <sup>st</sup> Replica 150 Bytes + {2 <sup>nd</sup> Replica 8 Bytes + 3 <sup>rd</sup> Replica 8 Bytes}	166 Bytes
Block 2	1 <sup>st</sup> Replica 150 Bytes + {2 <sup>nd</sup> Replica 8 Bytes + 3 <sup>rd</sup> Replica 8 Bytes}	166 Bytes
Total		482 Bytes





## Why HDFS Metadata to be maintained in RAM?

- Metadata Access from RAM is faster when compared to Disk
- Network bandwidth might be an issue if metadata is stored on different Host / NFS

## How to change NameNode's JVM size?

- `HADOOP_NAMENODE_OPTS` in `hadoop-env.sh`

```
HADOOP_NAMENODE_OPTS="$HADOOP_NAMENODE_OPTS -Xmx4g"
```

```
HADOOP_SECONDARYNAMENODE_OPTS=$HADOOP_NAMENODE_OPTS
```

## How to read HDFS Metadata?

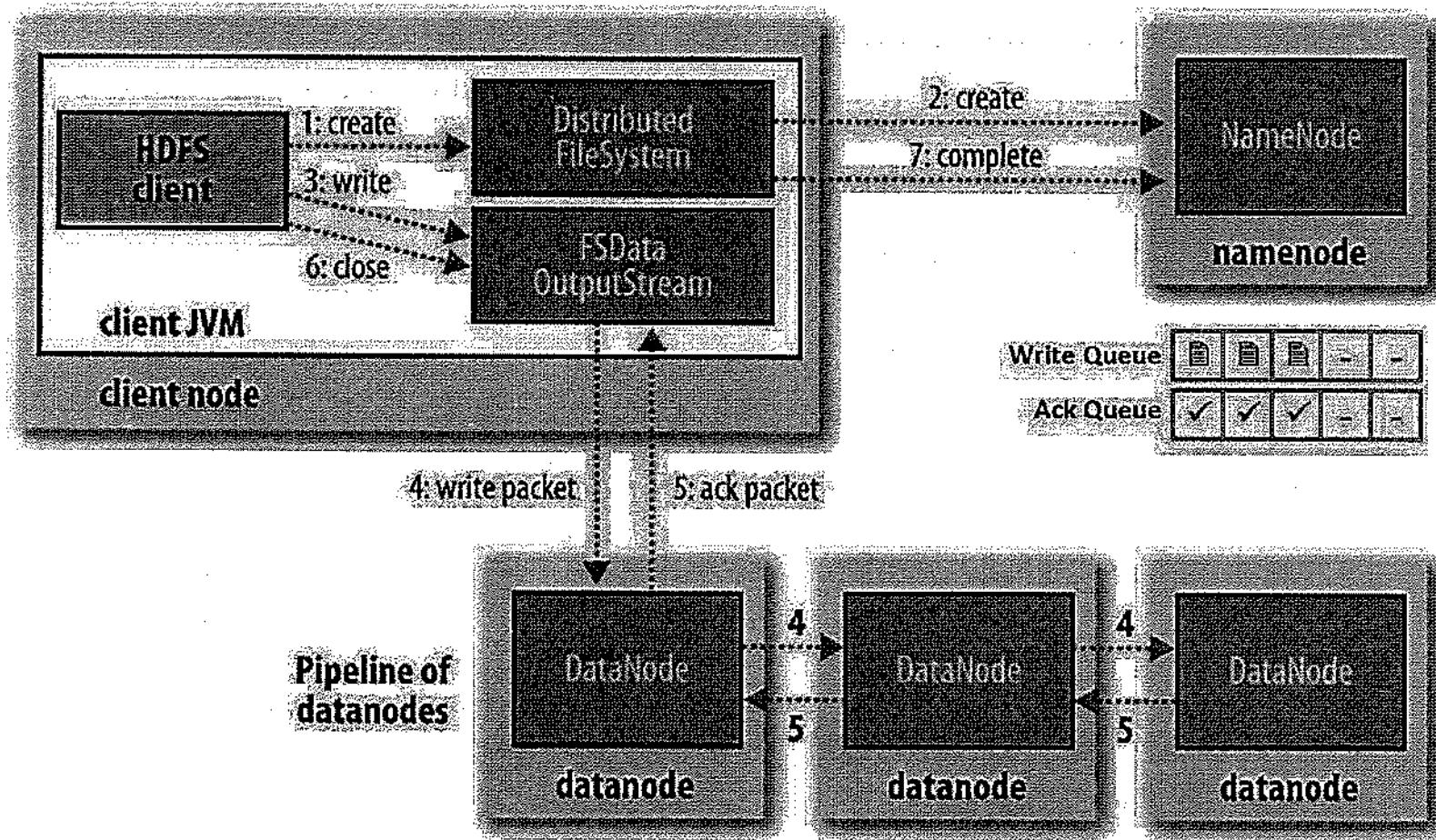
- Converting saved image to human readable file using

```
hadoop oiv -i fsimage -o fsimage.txt
```





## Anatomy of a File Write into HDFS





## Anatomy of a File Write into HDFS (cont'd)

- Client connects to the NameNode and submits the file write request
- NameNode performs various checks to make sure the file doesn't exist already and the Client has valid permissions to write the file
- If the checks are passed, NameNode creates an entry in *Edit Log* without any block numbers
- *DFSOutputStream* class will split the file into blocks and put into *WRITE/DATA* queue
- *DataStreamer* class will request the NameNode for the Block Name and list of DataNodes for each block and start sending the data packets to first DataNode
- As the data received by the first DataNode, it connects to the second DataNode and starts sending data. Second DataNode similarly connects to the third DataNode
- Once block writing is completed successfully, acknowledgement will be sent to *ACK Queue*
- *ACK* packets from the pipeline are sent back to the Client
- Client reports to the NameNode when all the blocks are written successfully and received *ACK*
- NameNode places the block name entries in its metadata and confirm the Client about successful file write





## What if a DataNode in the pipeline is failed?

- The pipeline will be closed and the failed packets in the **ACK** queue will be pushed back to **WRITE/DATA** queue
- NameNode provides a new pipeline with a set of good Nodes
- Data blocks will be written to the set of good Nodes in the pipeline
- As data blocks are written, Client calculates the checksum for each block and stores in **.meta** file in the DataNode. This will be used to ensure the integrity of the data when it is read later.
- NameNode marks write as successful if it meets the property **dfs.namenode.replication.min** (By default 1)
- Later, NameNode will understand that the block is under-replicated, and will re-replicate it to another DataNode



## Dealing with multiple Disks in Hadoop

**Logical Volume Manager (LVM):**

**Combination of separate disks maintained as single entity (`/data`)**

**Just a Bunch of Disks (JBOD):**

**Combination of separate disks maintained with comma(,) separation (`/mnt/disk1/data, /mnt/disk2/data, /mnt/disk3/data`)**

**NOTE:** With LVM, failure of a single disk causes the whole LVM inaccessible. But, if a disk failed in JBOD configuration, HDFS can continue to operate without the failed disk.

**Hadoop Settings:** Passing list of disks to `dfs.data.dir` parameter. Hadoop will use all disks that are available. When one disk goes offline, Hadoop will ignore that and continue with rest of the disks.

```
<property>
  <name>dfs.data.dir</name>
  <value>/mnt/disk1/data, /mnt/disk2/data, /mnt/disk3/data</value>
<property>
```



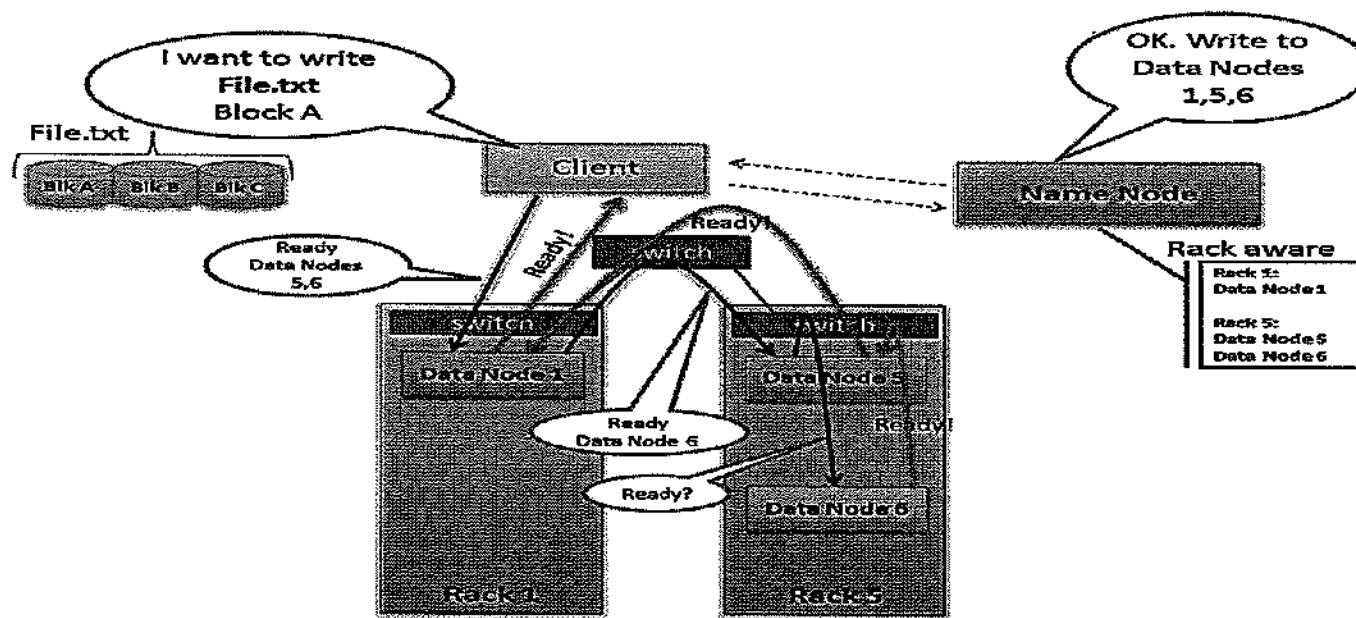
## Checking HDFS Status

- ***hdfs fsck* checks for missing or corrupted data blocks**
  - Unlike system *fsck*, does not attempt to repair errors
- **Can be configured to list all files**
  - Also all blocks for each file, all block locations, all racks
- **Example:**
  - `hdfs fsck / -files -blocks -locations -racks`
  - `hdfs fsck / -list-corruptfileblocks`
- **Good idea to run *hdfs fsck* on regular basis**
  - Choose a low-usage time of the Cluster to run this check
- **-move option moves corrupted files to */lost+found* folder on HDFS**
  - A corrupted file is one where all replicas of a block are corrupted
- **-delete option deletes corrupted files from HDFS**



## HDFS Block Replication Strategy (Rack Awareness)

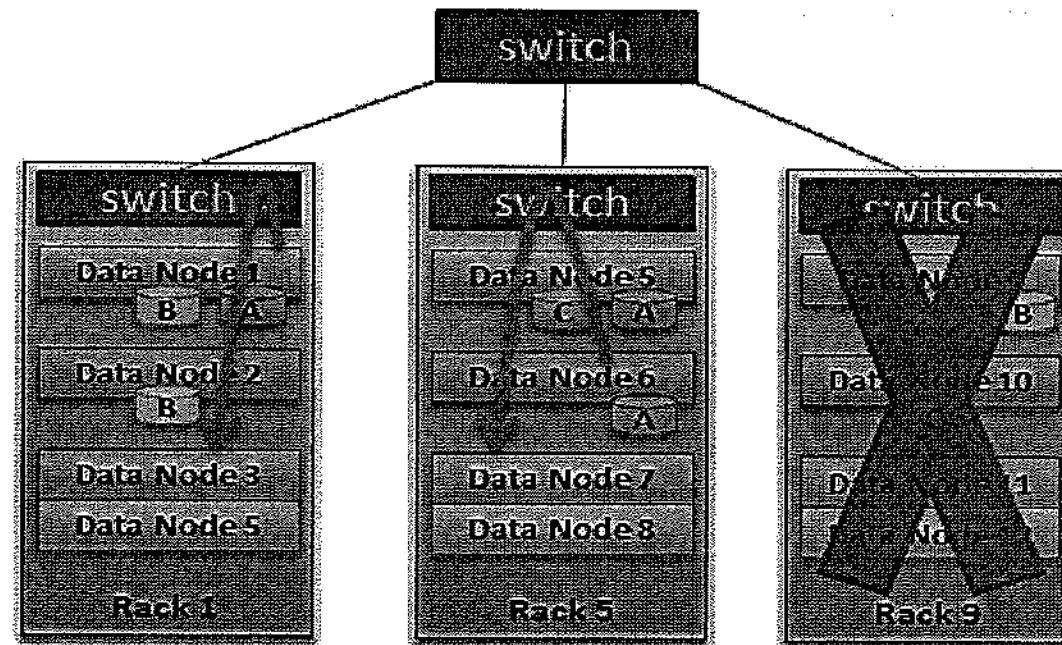
- First copy of the block is placed on a random DataNode with more free space and which is not too busy
- Second copy of the block is placed on a DataNode residing on a different rack
- Third copy of the block is placed on different DataNode in the same rack as the second copy





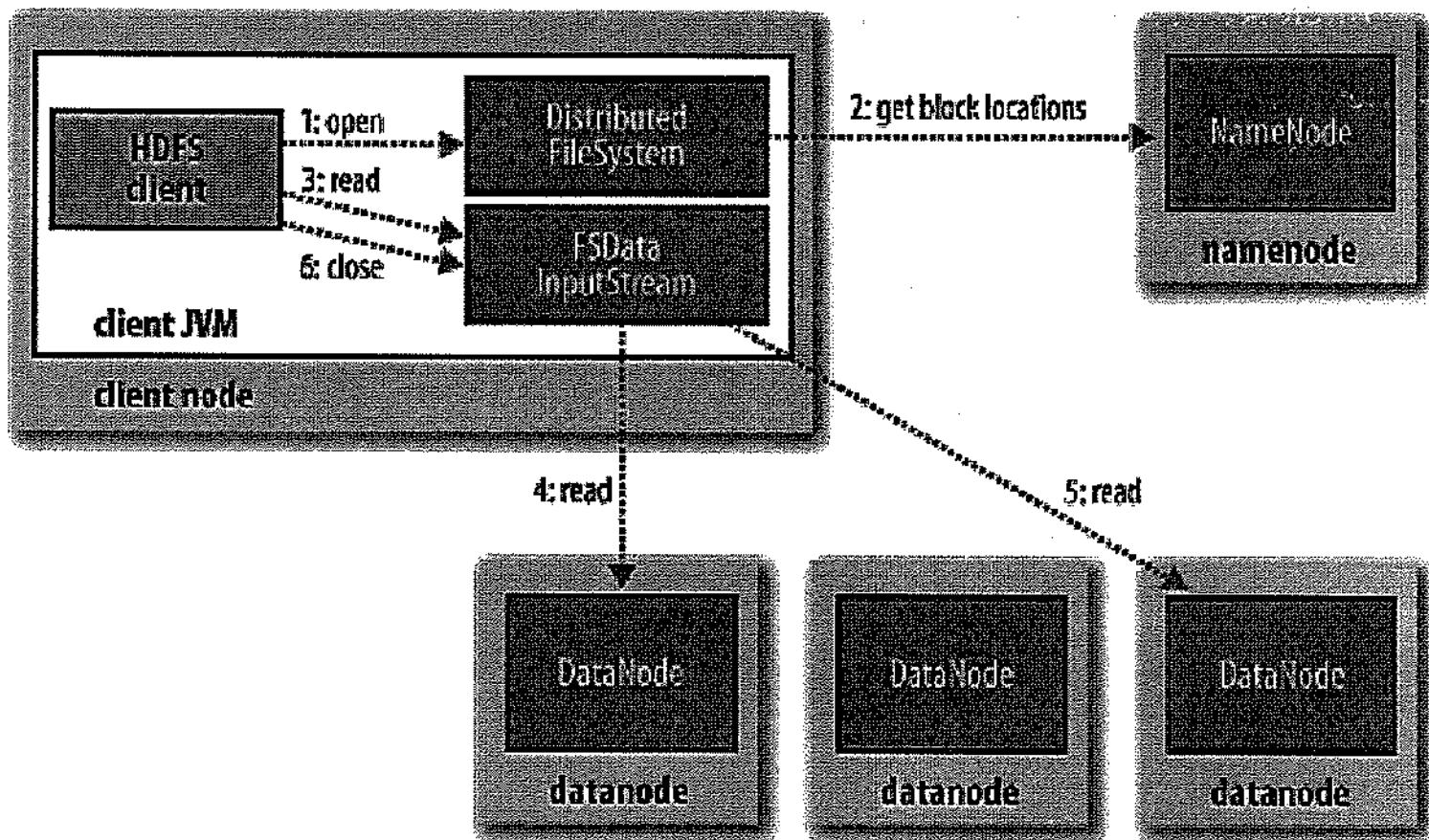
## Why Rack Awareness?

- Never lose all data if entire rack fails
- Assumption is that higher bandwidth and low latency in Rack





## Anatomy of a File Read from HDFS





## Anatomy of a File Read from HDFS (cont'd)

- Client connects to the NameNode and submits the file read request
- NameNode performs various checks to make sure the file is existing and the Client has valid permissions to read the file
- If the checks are passed, NameNode returns the name and locations of the file blocks from its metadata
- Client finds the closest DataNode in the list, and read the file blocks
- If DataNode is failed during the read process, Client will seamlessly connect to the next DataNode in the list and read the file block



## Dealing with data corruption (Data Integrity)

- As Client is reading the block, it also verifies the checksum
  - 'Live' checksum value of the block is compared with the checksum value which was stored in .meta file while creating the block
- If they differ, Client reads the file block from the next DataNode in the list
  - DataNode informs NameNode that a corrupted block has been found
  - NameNode will then delete the corrupted block and re-replicate that block elsewhere
- DataNode verifies the checksum value of all blocks on a regular basis to avoid '*bit rot*'
  - Can be checked using <http://datanode:50075/blockScannerReport>
  - Default is every *three weeks* after the block was created (*dfs.datanode.scan.period=504 Hrs.*)





## Dealing with data corruption (Setting up Trash)

This is similar to “*Recycle Bin*” in Windows. Its purpose is to prevent from unintentional deletion of the data from HDFS. Value is in minutes.

In *core-site.xml*:

*fs.trash.interval=360*

*fs.trash.checkpoint.interval=15*



**hdfs dfs -rm -skipTrash**      /path/to/file/to/delete (To ignore files moving to *.Trash* directory)

**hdfs dfs -expunge**      (Delete all the old checkpoints that are older than the “*fs.trash.interval*” value and create a new checkpoint of current *Trash* directory)

**hdfs dfs -rmr .Trash**      (Explicitly to delete the *.Trash* directory)





## Space Reclamation

- When a file is deleted by any user or application, it is immediately moved to a trash directory. Each user has their own trash directory under `/user/<username>/Trash`
- The file can be restored quickly as long as it remains in `.Trash`. Most recent deleted files are moved to the current trash directory (`/user/<username>/.Trash/Current`), and in a configurable interval, NameNode creates checkpoints (under `/user/<username>/Trash/<date>`) for files in current trash directory and deletes old checkpoints when they are expired.
- After the expiry of its life in `.Trash`, NameNode deletes the file from the HDFS namespace
- Trash feature is disabled by default. User can enable this feature by setting a value greater than zero for parameter `fs.trash.interval` in `core-site.xml`.
- Admin can configure an appropriate time to tell NameNode how often to create checkpoints in trash (the parameter stored as `fs.trash.checkpoint.interval` in `core-site.xml`), this value should be smaller or equal to `fs.trash.interval`.



## DataNode Failure, Heartbeats and Re-Replication

Each DataNode sends a Heartbeat to NameNode periodically. A network partition can cause a DataNode to lose its connectivity with the NameNode. NameNode detects this condition by the absence of a Heartbeat from DataNode. NameNode marks the DataNodes without recent heartbeats as dead and does not forward any new I/O requests to them. Any data that present in dead DataNode is not available to HDFS anymore. DataNode death may cause the replication factor of some blocks to fall below their specified value. NameNode constantly tracks which blocks need to be replicated and initiates the replication when required.

### When Re-Replication occurs?

- Block Replica is corrupted
- Hard disk on a DataNode is failed
- DataNode is unavailable for a certain amount of time
- Replication factor of a file is increased



## When DataNode marked as failed?

DataNode sends heartbeat to NameNode periodically. Due to any network glitch or service failure, DataNode may miss sending heartbeat to NameNode. NameNode will check below parameters to make sure DataNode is marked as dead.

***dfs.heartbeat.interval=3***

NN will wait for 10 iterations, i.e. 30 seconds

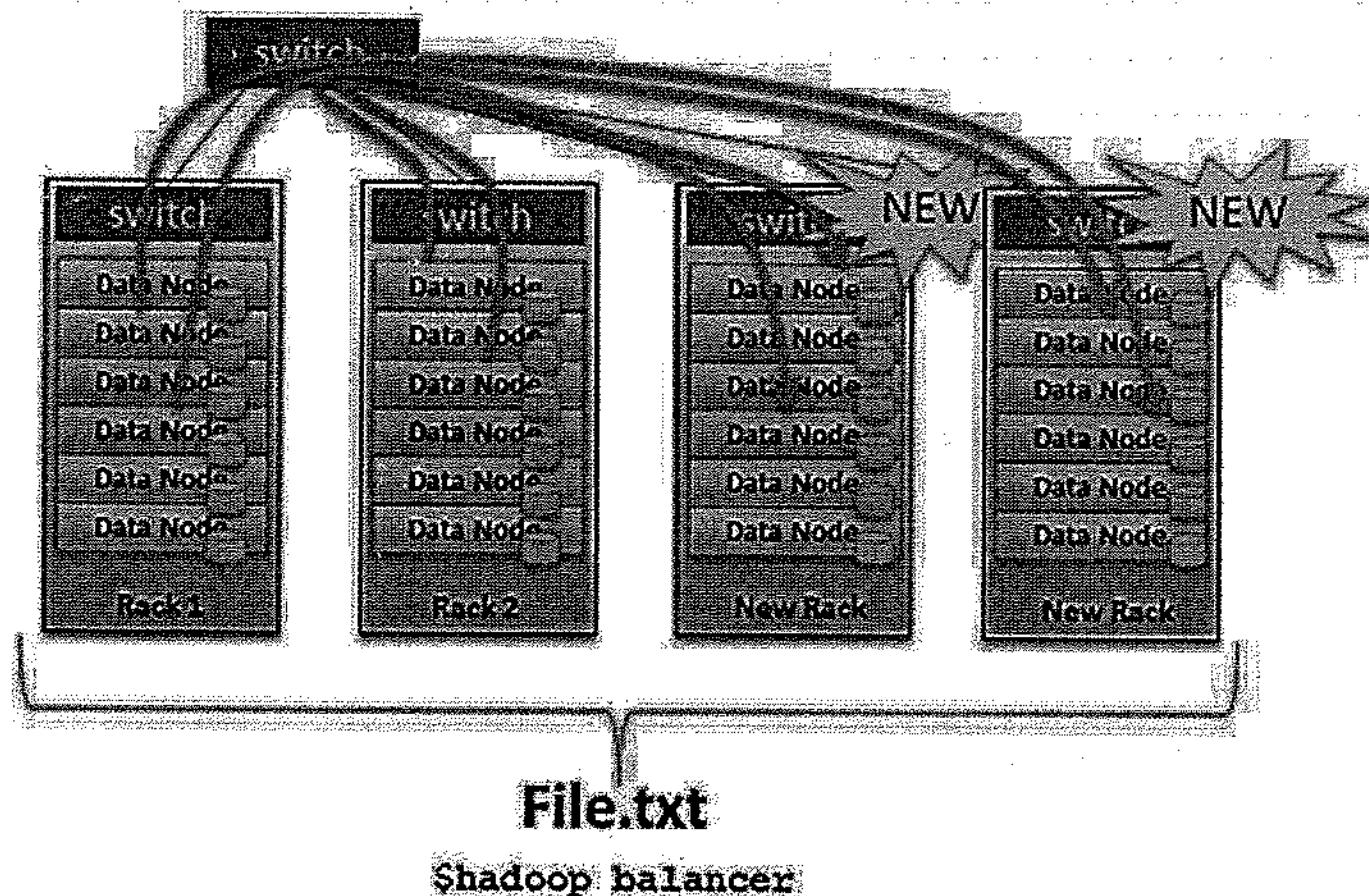
***dfs.namenode.heartbeat.recheck-interval=300000 milliseconds***

NN will wait for 2 iterations, i.e. 600 seconds

So, NN will mark the DN as dead after completion of ***10 minutes and 30 seconds***.



## Hadoop Cluster Rebalancing





## Cluster Rebalancing (Cont'd)

HDFS data might not always be placed uniformly across all the DataNodes. Most common reasons are addition of new DataNodes to the Cluster or deletion of old DataNodes from the Cluster.

NameNode analyzes block placement and rebalances data across all the DataNodes in the Cluster.

While rebalancing the blocks, NameNode considers all block placement algorithms before choosing the DataNodes to receive the blocks and make sure HDFS data spread uniformly across all the DataNodes in the Cluster.

*start-balancer.sh*

(By default, 10% of RAM & 1MB per second)

*hdfs balancer -threshold 10*

(Percentage of disk capacity)

*hdfs dfsadmin -setBalancerBandwidth 20000000*

(only move 20000000 bytes per second)





## Change of Replication Factor

When the replication factor of a file is reduced, NameNode selects excess replicas that can be deleted. NameNode transfers this information to the respective DataNode in the next heartbeat. DataNode then removes the corresponding blocks and the corresponding free space appears in the Cluster. Once again, there might be a time delay between the completion of the setReplication API call and the appearance of free space in the Cluster.

When the replication factor of a file is increased, NameNode transfers this information to the DataNode in the next heartbeat. DataNode starts replicating the corresponding blocks in the Cluster. Once again, there might be a time delay in the completion block replication in the Cluster.

To see replication factor of a file:

```
hdfs dfs -stat "%n %o" /path/to/file
```

To change replication of a file:

```
hdfs dfs -setrep 2 /path/to/file
```





## Change of Block Size

If the block size is changed,

- New block size will be applicable only for the new files
- Existing files on HDFS will have same old block size
- Old file's block size can be changed as below.

*hdfs dfs -Ddfs.block.size=XX -cp /path/to/old/files /path/to/new/files/with/large/block/size*

If block size is increased,

- HDFS will have fewer blocks which will reduce metadata size
- Since data is stored in fewer blocks, it can reduce total throughput for parallel access
- It will be more difficult to YARN to schedule data-local tasks.

If block size is decreased,

- HDFS will have more blocks which will increase metadata size and decrease NN performance
- Since data is stored in more blocks, it can increase total throughput for parallel access
- It will be more difficult to YARN to manage resource allocation and job monitoring





## File Systems supported by Hadoop

File System	URI scheme	Java implementation	Description
Local	file	fs.LocalFileSystem	A filesystem for a locally connected disk with Client-side checksums. Use RawLocalFileSystem for a local filesystem with no checksums.
HDFS	hdfs	hdfs.DistributedFileSystem	Hadoop's distributed filesystem. HDFS is designed to work efficiently in conjunction with MapReduce.
HFTP	hftp	hdfs.HftpFileSystem	A filesystem providing read-only access to HDFS over HTTP. (Despite its name, HFTP has no connection with FTP.) Often used with distcp to copy data between HDFS Clusters running different versions.
WebHDFS	webhdfs	hdfs.web.WebHdfsFileSystem	A filesystem providing secure read-write access to HDFS over HTTP. WebHDFS is intended as a replacement for HFTP and HSFTP.
HAR	har	fs.HarFileSystem	A filesystem layered on another filesystem for archiving files. Hadoop Archives are typically used for archiving files in HDFS to reduce NameNode's memory usage.





## File Systems supported by Hadoop (cont'd)

Filesystem	URI scheme	Java implementation	Description
KFS (Cloud-store)	kfs	fs.kfs.KosmosFileSystem	CloudStore (formerly Kosmos filesystem) is a distributed filesystem like HDFS or Google's GFS, written in C++.
FTP	ftp	fs.ftp.FTPFileSystem	A filesystem backed up by an FTP server.
S3 (native)	s3n	fs.s3native.NativeS3FileSystem	A filesystem backed up by Amazon S3
S3 (block based)	s3	fs.s3.S3FileSystem	A filesystem backed up by Amazon S3, which stores files in blocks (much like HDFS) to overcome S3's 5 GB file size limit.
View	viewfs	viewfs.ViewFileSystem	A Client-side mount table for other Hadoop filesystems. Commonly used to create mount points for federated NameNodes





## Compression formats supported by Hadoop

Format	Tool	Algorithm	Extension	Splittable	Hadoop Compression Codec
DEFLATE <sup>a</sup>	N/A	DEFLATE	.deflate	No	org.apache.hadoop.io.compress.DefaultCodec
Gzip	gzip	DEFLATE	.gz	No	org.apache.hadoop.io.compress.GzipCodec
bzip2	bzip2	bzip2	.bz2	Yes	org.apache.hadoop.io.compress.BZip2Codec
LZO	lzo	LZO	.lzo	No <sup>b</sup>	com.hadoop.compression.lzo.LzopCodec
Snappy	N/A	Snappy	.snappy	No	org.apache.hadoop.io.compress.SnappyCodec

**a - 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 footer.) The .deflate filename extension is a Hadoop convention.

**b -** However, LZO files are splittable if they have been indexed in a preprocessing step.



## How GZIP works:

GZIP provides a lossless compression, that is, we can recover the original data when decompressing it. It is based on the *DEFLATE* algorithm, which is a combination of LZ77 and Huffman coding.

The LZ77 algorithm replaces repeated occurrences of data with references. Each reference has two values: back pointer of <distance, length>. Let's see an example:

*Original text: "ServerGrove, the PHP hosting company, provides hosting solutions for PHP projects"* (81 bytes)

*LZ77: "ServerGrove, the PHP hosting company, p<32,3>ides<26,9>solutions for<52,5><35,3>jects"* (73 bytes, assuming that each reference is 3 bytes)

Huffman coding is a variable-length coding method that assigns shorter codes to more frequent "characters".

*Huffman: "1110 00 01 10 00 01 1111 01 110 10 00"* (27 bits)

GZIP will delete the input file and place only the .gzip as output.





## How LZO works?

LZO is a file compressor which is very similar to *gzip*. LZO uses the *LZO data compression library* for compression service and its main advantages over *gzip* are much higher compression and decompression speed (at the cost of some compression ratio).

LZO does not delete the input file(s) by default.

## How SNAPPY works?

Snappy (previously known as Zippy) is a fast data compression and decompression library written in C++ by Google based on ideas from LZ77. It does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression. Compression speed is 250 MB/s and decompression speed is 500 MB/s using a single core of a *Core i7* processor running in *64-bit mode*. The *compression ratio* is 20–100% lower than *gzip*.





## Comparison of Compression formats

Algorithm	Compression Ratio	IO Performance Increase
GZIP	28%	80%
BZIP	21%	50%
Snappy	40%	25%
LZF	40%	21%
LZO	41%	5%
ZLIB	48%	-16%

**Compression Ratio:** The size reduced after compressing the file.

Ex: 100MB file will have the size of 60MB after compression. 40MB has been reduced.

**IO Performance Increase:** The time reduced to read the compressed file.





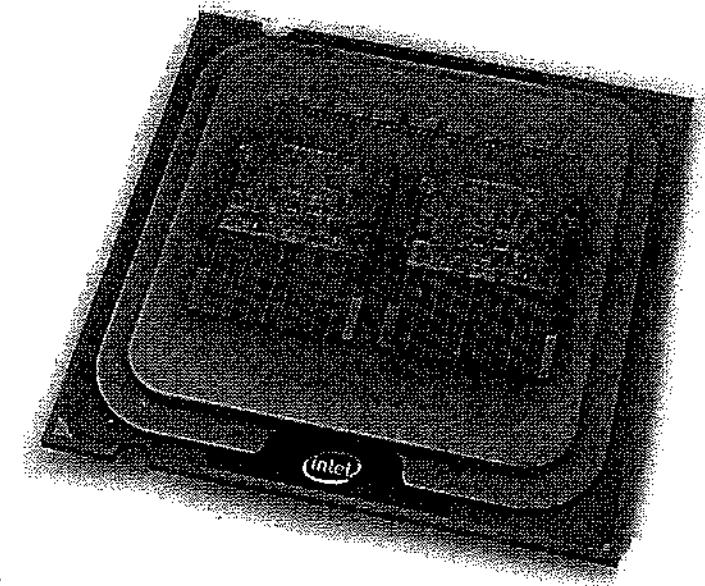
# Processing Layer (MR & YARN) and it's internals ..





## CORE

Core is usually the basic computation unit of the CPU. It can run single/multiple threads based on *hyper threading* feature. One thread can launch one JVM which is equivalent to one Mapper or Reducer in Hadoop.



## CPU

CPU may have one or more cores to perform tasks at a given time. Total number of tasks performed by a CPU are *number cores \* number of HW-threads per core*.

Ex: 1 CPU with 2 Cores with hyper threading of 2.

Total No. of tasks = 1 CPU \* 2 Cores \* 2 Threads = 4 Tasks





## Java Virtual Memory (JVM) Internals

Heap Area	Method Area	Java VM Stack	Native Method Stack	PC Registers
Runtime data area for the variables, arrays, objects etc. used in your program	Stores per-class structures such as methods and constructors used in class, instance, and interface initialization	Hold local variables and partial results and plays a part in the method invocation and return.	Support native methods (methods written in a language other than the Java programming language)	Each of the JVM threads has its own program counter (pc) register. At any point, each of the JVM threads is executing the code of a single method, namely the current method for that thread.



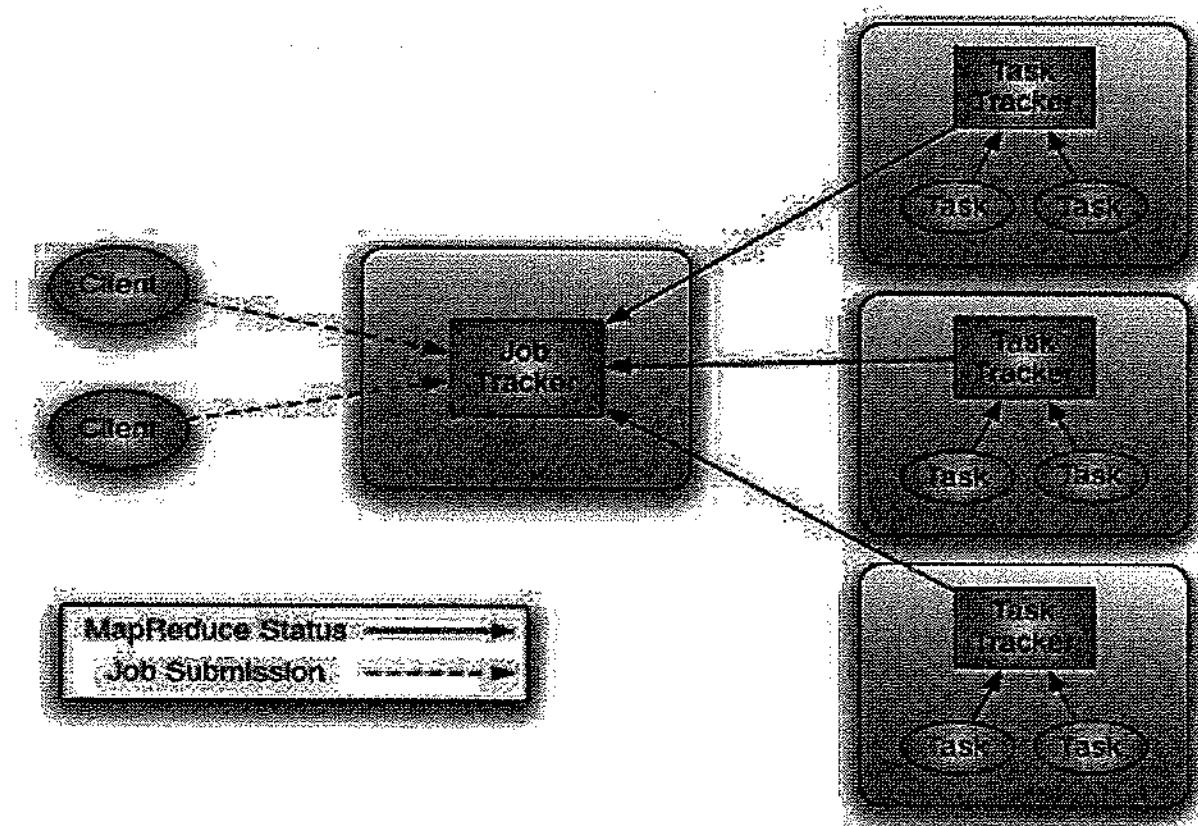


## Map Reduce (High Level Architecture)

It's a programming model, which is independent of the programming language or platform.

It has 2 phases:

- Map
- Reduce





## Job Tracker (JT)

- **Master Node in MapReduce framework**
- **First point of contact for all processing layer requests**
- **Keeps track of all *Task Trackers* and the resources they have**
- **Maintains the life cycle of the job till it is completed**
- **Job status will be acknowledged to the Client**
- **Job Tracker is responsible for “Resource Management” and “Job Scheduling”**



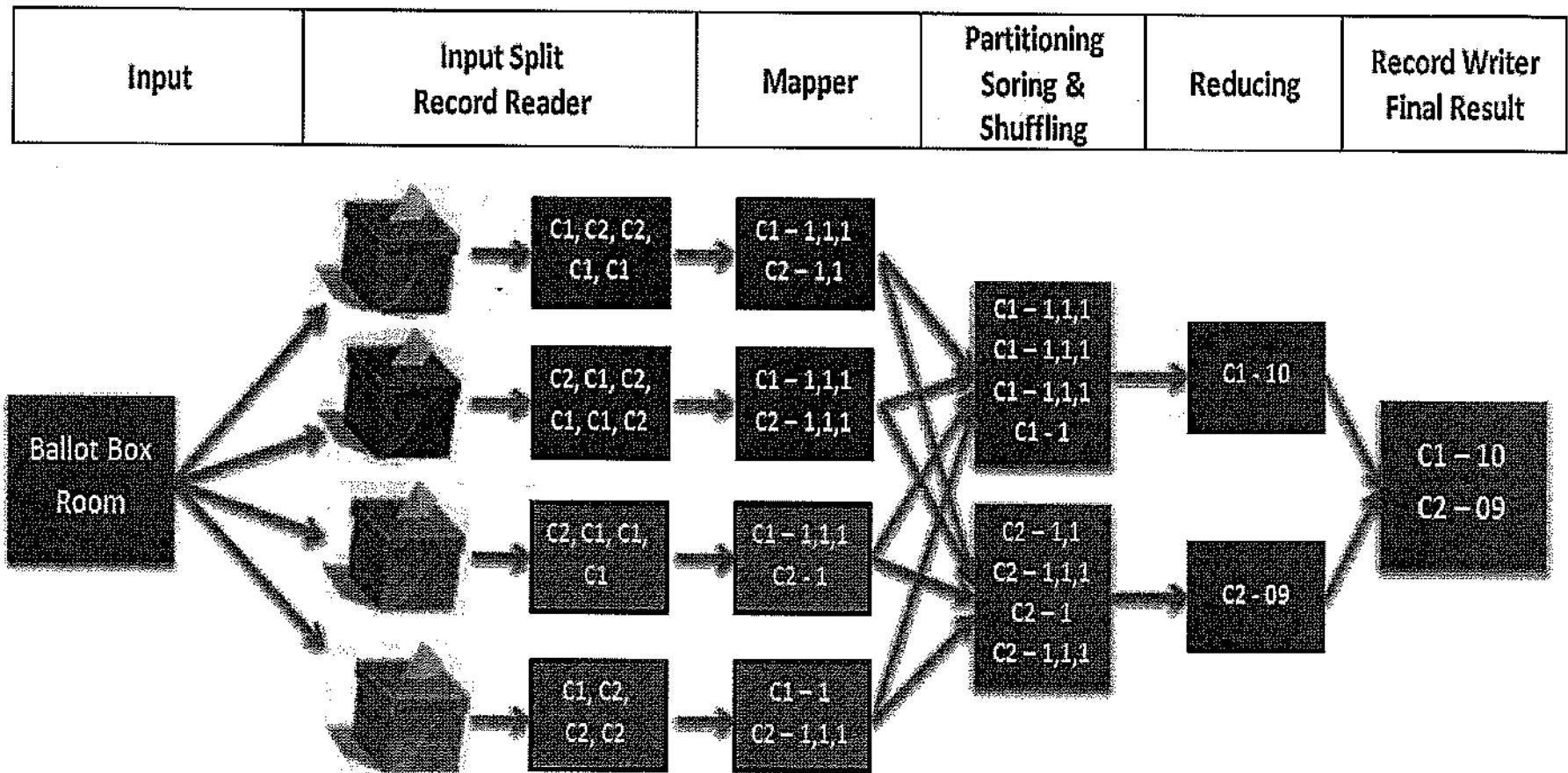
## Task Tracker (TT)

- **Slave Node in MapReduce framework**
- **An instance of the Task Tracker daemon runs on every DataNode**
- **Receives processing requests from Job Tracker**
- **Run each map/reduce task in separate JVM**
- **Responsible to update the status of each Map/Reduce task to Job Tracker**

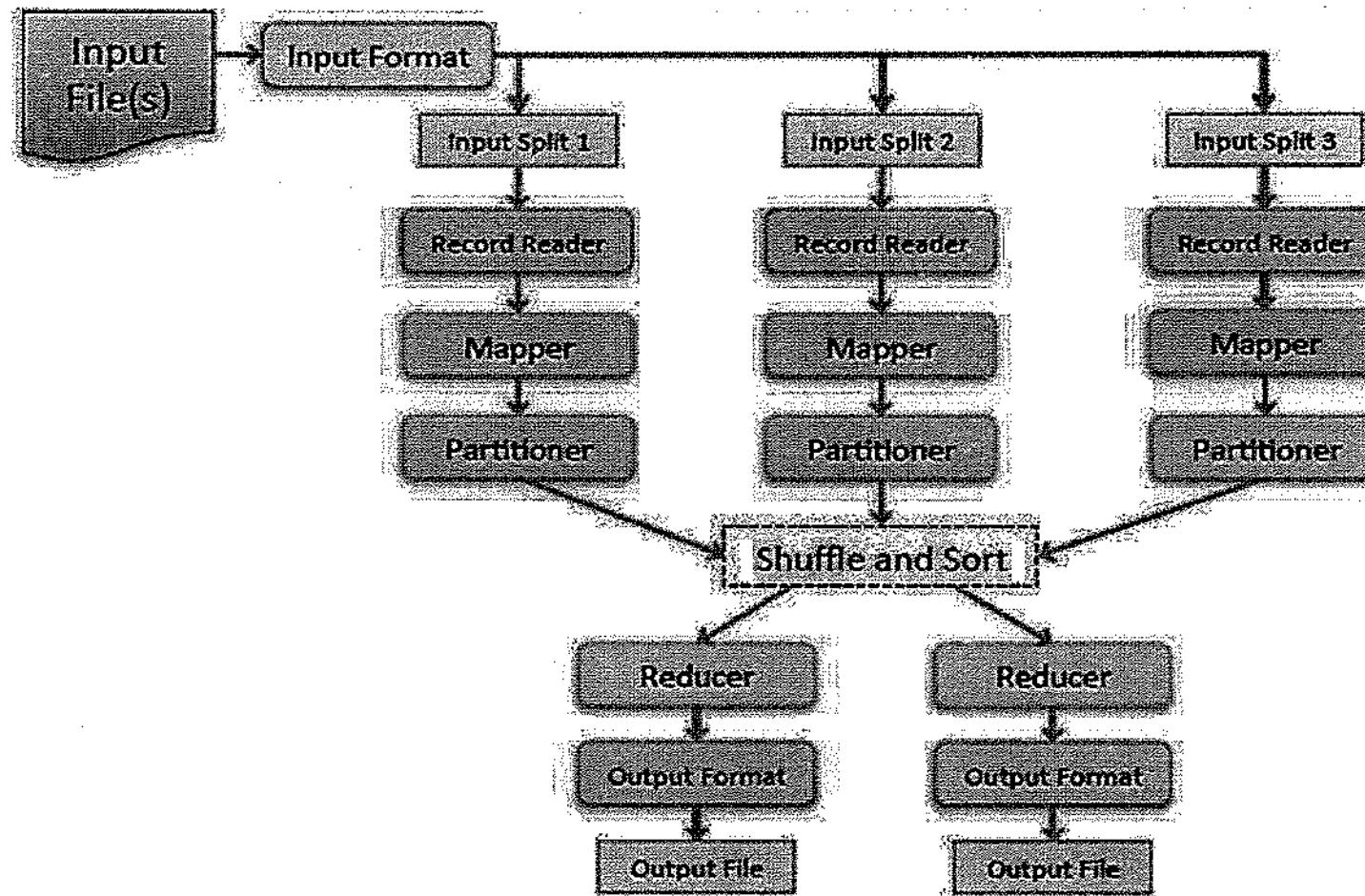




## How does MapReduce work?

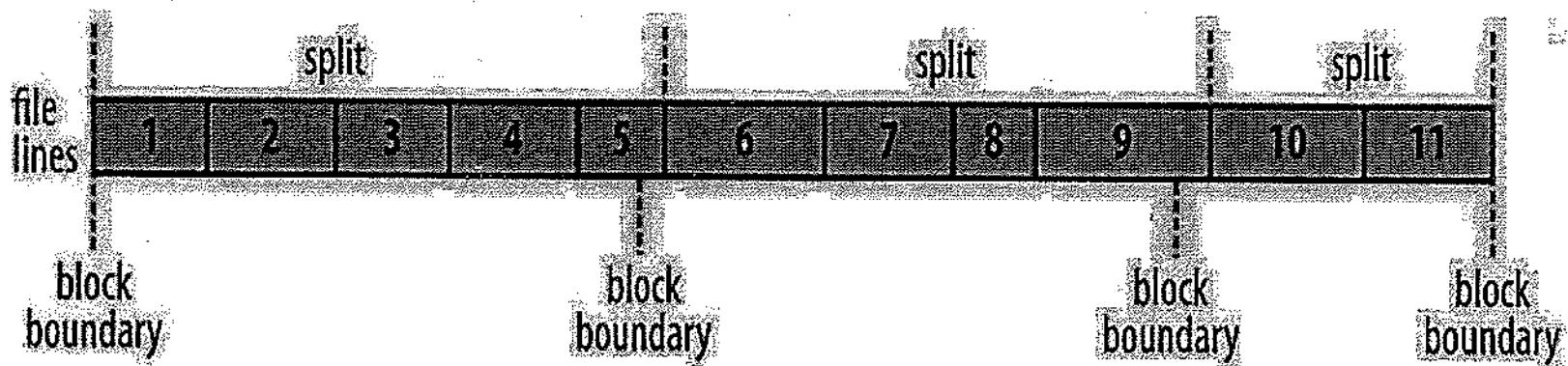


## MapReduce Pipeline



## Relationship between Input Split and HDFS Block

- A file on HDFS will be divided into blocks
- Records can be split across HDFS blocks
- Split is a logical block which honors complete record boundaries
- *Split size will be defined by *FileInputFormat**





## Anatomy of MapReduce Job Submission

- Client submits the job to Job Tracker and Job Tracker provides a job ID to the Client as a reference number
- Job Tracker contacts NN and get location of the file blocks to be processed as part of the job
- Job Tracker locates Task Tracker Nodes with available resources at or near the data blocks
- Job Tracker submits the task to Task Tracker and monitor the task status
- Task Tracker will notify Job Tracker when a task is completed or failed
- If task is failed, Job Tracker will submit the failed task to other Task Tracker where the next block replica is available. If the task is still failing, JT will submit the task to some other Task Tracker. A task can be submitted maximum of *4 times* by default.
- If any task fails for more than 4 times, Job Tracker marks entire job as failed
- When all tasks are completed successfully, Job Tracker updates its status to Client
- Client can get the job status from Job Tracker during job execution





## Running simple MR Jobs in Hadoop

- MR job will place the data in the given output folder
- If the output folder is already existing, MR Job will fail
- Every Hadoop version comes with few example jars

Ex:

```
hadoop jar /usr/local/hadoop/hadoop-examples-1.2.0.jar wordcount /sample.txt /output
```



## How the tasks will be submitted to Task Trackers?

- Job Tracker gets the block locations from NameNode
- Job Tracker will copy the necessary files (usually JARs) to the Task Trackers before any tasks for the job are executed on that Nodes
- *Distributed Cache* is a facility provided by MR framework to cache the necessary files needed for job execution
- *Distributed Cache* can be used to distribute simple, read-only data/text files and more complex types such as archives and jars. Archives (zip, tar, tgz and tar.gz files) are *un-archived* at Task Tracker Nodes.
- These files should have the execute permissions.





## Choosing the number of Mappers & Reducers

### *Mappers:*

- The number of mappers is usually driven by the number of Input Splits
- Thumb rule is to select a Mapper which runs for a max of *1-2 minutes*

### *Reducers:*

- Default 1 Reducer for a job. Job will be very slow if single Reducer is used as all the intermediate data flows through single Reducer
- Real world jobs require more number of Reducers
- Increasing Reducers makes the job execution easy, but ends up with lots of small files
- The number of Reduce tasks can also be increased manually using below properties
  - `conf.setNumReduceTasks(int num)`
  - `-D mapred.reduce.tasks=2`
- Thumb rule is to select a Reducer which runs for a max of *5 minutes* and produce at least 1 HDFS block of data





## Controlling maximum length of a record

- A Record can be more than a block size (Ex: 2GB)
- Task may fail if the Split size is more than the memory assigned to Mapper
- This can be controlled by setting up `mapreduce.input.linerecordreader.line.maxlength`
- If this parameter is set, Record Reader will skip the records which are longer than this size without failing the task
- Better if this parameter is job specific instead of common to all jobs





## Commissioning (Adding) Nodes to the Cluster

In *hdfs-site.xml*:

```
<property>
    <name>dfs.hosts.include</name>
    <value>/usr/local/hadoop/conf/include.txt</value>
</property>
```

- Update the new server IP address in '*include.txt*' file
- Start DataNode & Task Tracker services in the new Nodes
- Refresh NameNode: *hadoop dfsadmin -refreshNodes*
- Refresh Job Tracker: *hadoop mradmin -refreshNodes*
- Crosscheck whether the new Nodes are successfully registered
- Add the new Nodes to the '*slaves*' file
- Run balancer to move HDFS blocks to new DataNodes





## De-Commissioning (Removing) Nodes from the Cluster

In *hdfs-site.xml*:

```
<property>
    <name>dfs.hosts.exclude</name>
    <value>/usr/local/hadoop/conf/exclude.txt</value>
</property>
```

- Update the old server IP address in the '*exclude.txt*' file
- Remove the old server IP address from '*include.txt*' file, if exists
- Refresh NameNode: *hadoop dfsadmin -refreshNodes*
- Refresh Job Tracker: *hadoop mradmin -refreshNodes*
- Crosscheck for the successful removal of Nodes "*Decommissioned*" & "*Decommission in Progress*"
- Remove the Nodes from the '*slaves*' file

**NOTE:** Even if we restart the services on excluded Nodes, NameNode will keep killing those services till we clear the "*exclude.txt*" file.





## Limitations of Job Tracker

- Single Point of Failure
- Suitable for *Batch Processing* only
- Heavily loaded with
  - Resource Management
  - Job Scheduling
  - Job Monitoring
- Non-scalability and inefficiency (Limited to 4,000 Nodes and 40,000 Tasks)
- No Job Tracker *High Availability* in Hadoop 1.X
- No Log Aggregation

## Hadoop 2.X Setup differences in Processing Layer

- Job Tracker / Task Tracker have been upgraded to Resource Manager / NodeManager
- There is no “*hadoop-daemon.sh start resourcemanager*” command, and it is upgraded to “*yarn-daemon.sh start resourcemanager*”
- The job execution is done by YARN framework





## Yet Another Resource Negotiator (YARN) Components

**Resource Manager (RM)** – Controls the container usage on the Cluster

- **App Manager (AMg)** – Manage Application Masters across the Cluster
- **Job Scheduler** – Responsible for allocating containers to the submitted jobs

**Application Master (AMa)**

- One Application Master for one application
- Responsible for negotiating the containers with RM
- Monitors the progress of the containers and tracks the status of the application

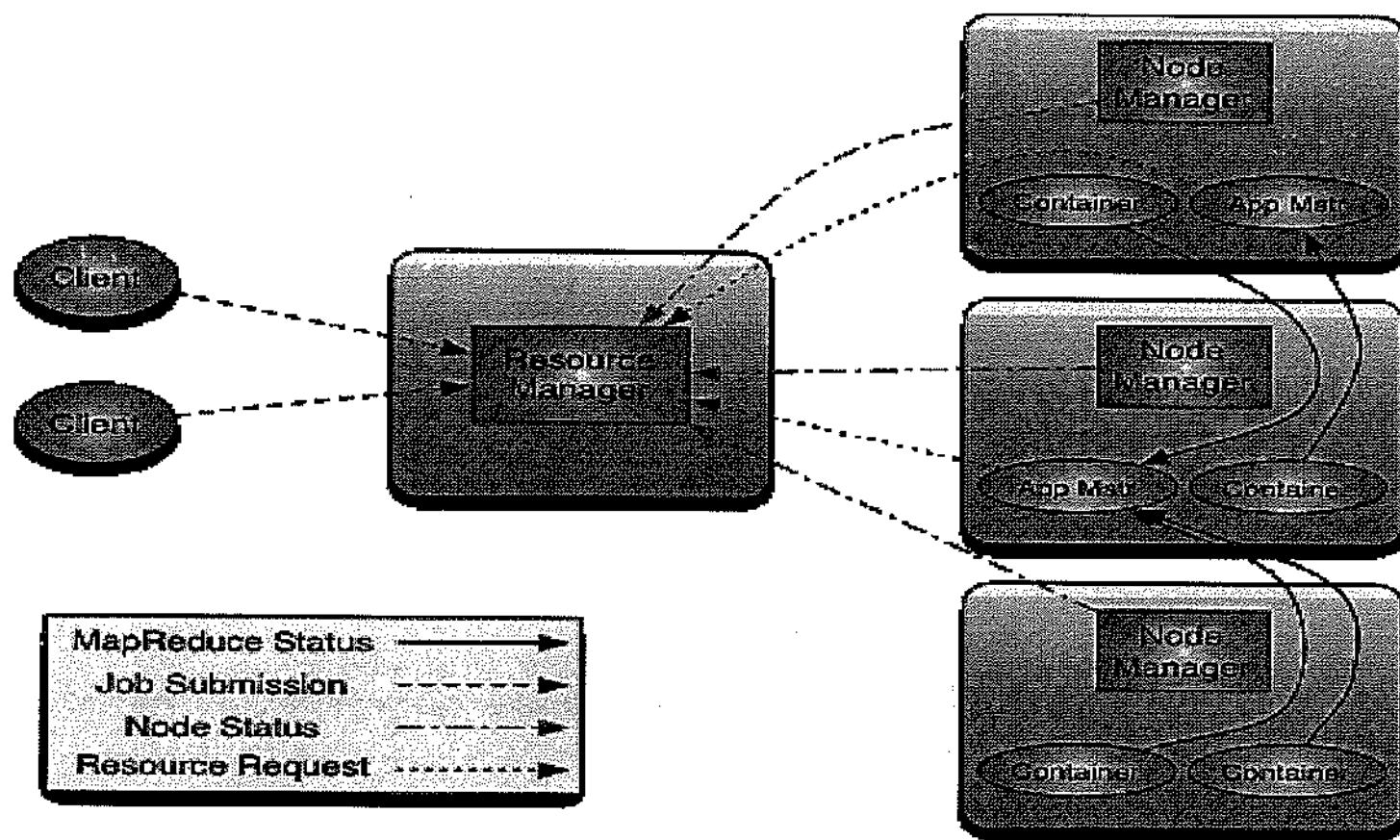
**Node Manager (NM)**

- Similar to Task Tracker, runs on each DataNode and takes care of execution of containers





## YARN Architecture





## Resource Manager's Responsibilities

- Manages the NodeManager(s) with the help of heartbeat from NodeManager(s)
- Instructs the NodeManager to launch Application Master containers and monitor the same
- Handles Application Masters' requests for new containers to run the tasks
- Determines how the resources are allocated with the help of YARN Schedulers
- Deallocates containers when they expire or when the application gets completed
- Manages Cluster level security
- Maintains Cluster State Store and Application State Store

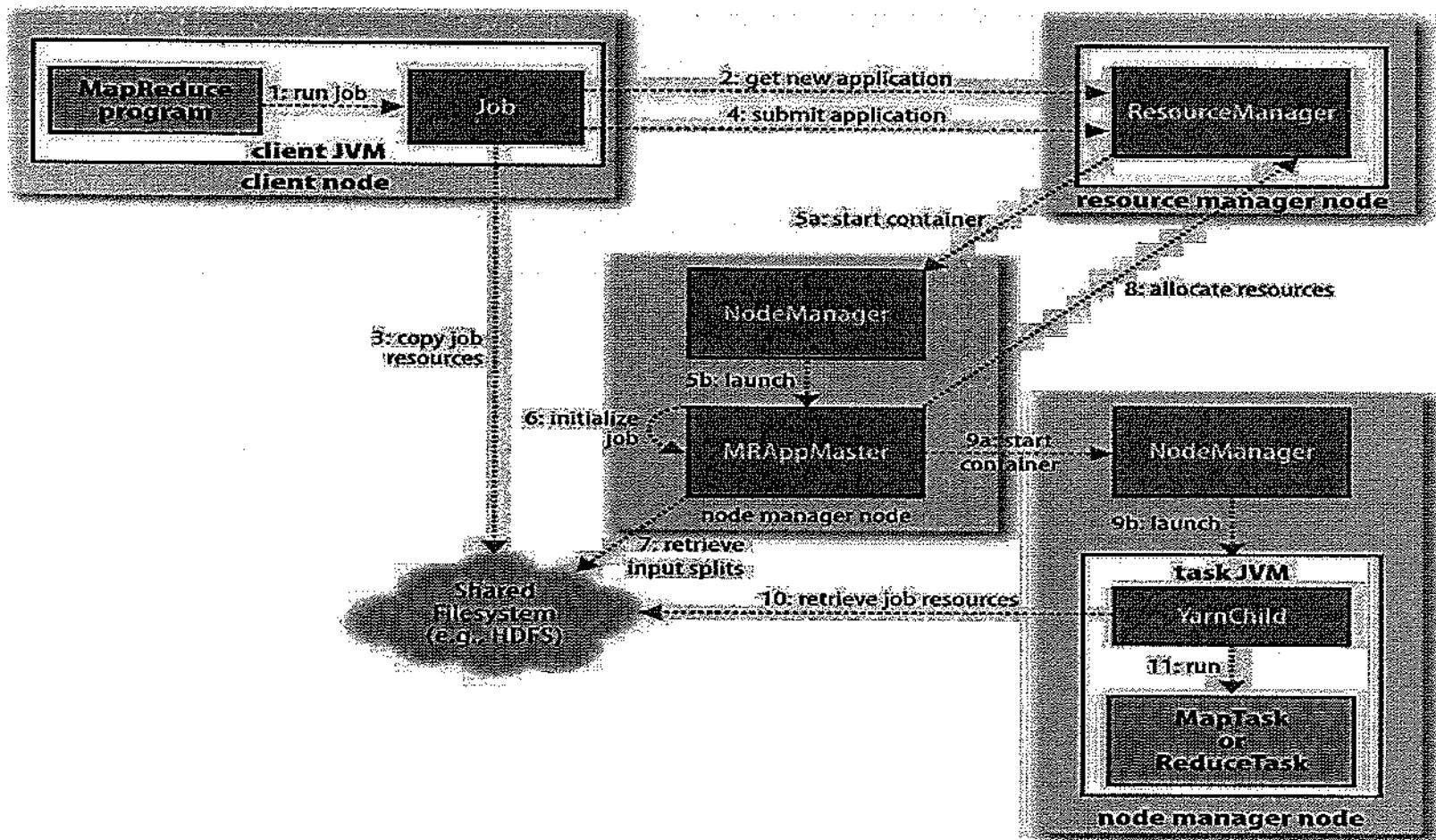
## Node Manager's Responsibilities

- Registers with RM and provide info on available resources and container status thru heartbeat
- Launch Application Masters on request from the Resource Manager
- Launch tasks into containers on request from Application Masters
- Monitor resource usage by containers; kill runaway processes
- Aggregate logs for an application and save them to HDFS





## Job Submission in YARN





## Job Submission in YARN (cont'd)

- Client submits the job/application to Resource Manager and Resource Manager provides an application ID to the Client as a reference number
- RM chooses any NM with available resources and launches new Application Master (AMa) and assigns MR job to it
- AMa contacts NameNode and get location of the file blocks to be processed as part of the job
- AMa identifies the required number of resources to process the job and negotiates with Resource Manager to allocate the same
- RM locates Node Manager nodes with available resources at or near the data blocks and allocate the resources
- AMa launch YARN child JVMs (containers) to run mapper/reducer tasks
- AMa is responsible for end-to-end execution of all tasks
- AMa will be updating the tasks' status to Resource Manager & Client on frequent intervals
- Once all tasks are successfully completed, AMa sends the job information and logs to Job History Server
- Resource Manager releases all the resources (containers) held by the application and reuse them for another application





## Memory Allocation in YARN

**yarn.nodemanager.resource.memory-mb**  
(Node Manager)

**YARN container**  
**cap by** `yarn.scheduler.maximum-allocation-mb`

**Mapper/Reducer**  
**cap by**  
`mapreduce.map.memory.mb`  
`mapreduce.reduce.memory.mb`

**JVM**  
**cap by**  
`mapreduce.map.java.opts`  
`mapreduce.reduce.java.opts`





## Advantages of YARN framework

- Flexible Resource Model with multiple Data Processing Algorithms
- High Availability for components
- Scalability and High Cluster Utilization
- Log Aggregation





# Hadoop Job / Application Internals.



## Job/Application ID, Task ID, and Task Attempt IDs

- Job IDs are generated by Job Tracker (in 1.X)
- Application IDs are generated by Resource Manager (in 2.X/3.X)

Ex: *job\_1410450250506\_0003 / application\_1410450250506\_0003*

- *1410450250506* represents start time of Job Tracker/Resource Manager
- *0003* represents the job counter
- Task IDs are formed by replacing the *job* prefix of a job with a *task* prefix and adding a suffix to identify the task within the job.

Ex: *task\_1410450250506\_0003\_m\_000002*

- *000002* is the third map(m) task of the job *job\_1410450250506\_0003*
- Tasks may be executed more than once, due to task failure or speculative execution, so to identify different instances of a task execution, task attempts are given unique IDs.

Ex: *attempt\_1410450250506\_0003\_m\_000002\_0*

- *0* is the first attempt of the task *task\_1410450250506\_0003\_m\_000002*
- Task attempts are allocated during the task run as needed





## Counters in Hadoop

- Useful channel to collect statistics about the job
- Useful for problem diagnostics and quality control
- Can be tracked in web UI of Job Tracker or Resource Manager
- *File System Counters* are maintained by each Task Attempt
- *Job Counters* are maintained by Job Tracker / Application Master
- *Map Reduce Counters* are maintained by Job Tracker / Application Master
- Other Job specific counters



## Counters in Hadoop (cont'd)

Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	3163	0	3163
	FILE: Number of bytes written	91098	0	91098
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	MAPRED: Number of bytes read	3134	0	3134
	MAPRED: Number of bytes written	6590	0	6590
	MAPRED: Number of large read operations	0	0	0
	MAPRED: Number of read operations	8	0	8
Job Counters	MAPRED: Number of write operations	69	0	69
	Name	Map	Reduce	Total
	DISK_MILLIS_MAPS	0	0	1539
	Launched map tasks	1	0	1
	Rack-local map tasks	1	0	1
	Total megabyte-seconds taken by all map tasks	0	0	7879680
	Total time spent by all map tasks (ms)	0	0	3078
	Total time spent by all mappers in corrupted slots (ms)	0	0	9234
	Total vcore seconds taken by all map tasks	0	0	3076
Map-Reduce Framework	Name	Map	Reduce	Total
	CPU time spent (ms)	420	0	420
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	0	0	0
	InputSplitBytes	197	0	197
	Map input records	33	0	33
	Map output records	0	0	0
	Merged Map outputs	0	0	0
	Physical memory (bytes) snapshot	306880512	0	306880512
	Spilled Records	0	0	0
	Total committed heap usage (bytes)	2022703104	0	2022703104
	Virtual memory (bytes) snapshot	3086266368	0	3086266368
com.aexp.cornerstone.diparser.FixedWidthParserCounters	Name	Map	Reduce	Total
	ERROR_RECORDS	0	0	0
	GOOD_RECORDS	31	0	31
File Input Format Counters	Name	Map	Reduce	Total
	Bytes Read	2937	0	2937
File Output Format Counters	Name	Map	Reduce	Total
	Bytes Written	0	0	0



## Job History

**Job history refers to the events and configuration for a completed MapReduce job. It is retained regardless of whether the job was successful or failed, in an attempt to provide useful information for the user running a job.**

**Job history files are stored in HDFS by Application Master, in a directory set by the *mapreduce.jobhistory.done-dir* property. Job history files are kept for *one week* (by default) before being deleted by the framework.**

**The history log includes job, task, and attempt events, all of which are stored in a file in JSON format. The history for a particular job may be viewed through the web UI for the job history server (which is linked to from Resource Manager page) or via the command line using *yarn logs -applicationId <application ID>* (which you point at the job history file).**



## Logging & Log Files

Logs are very important to track how things work and to troubleshoot when something is wrong.

Each Daemon in Hadoop writes logs and we can control what they write/log.

*Logging levels*

WARN, INFO, DEBUG, ERROR

*Logging*

log4j.properties in *conf* directory

This property can be checked by <http://resource-manager-host:8088/logLevel>





## Hadoop Daemon Logs (for Administrators)

**Daemon Logs** are created by the Hadoop daemons, and exist on all machines running at least one Hadoop daemon. Some of the files end with `.log`, and others end with `.out`. The `.out` files are only written to when daemons are starting. After daemons have started successfully, the `.out` files are truncated. By contrast, all log messages can be found in the `.log` files, including the daemon start-up messages that are sent to the `.out` files.

## Job Configuration XML (for Users)

The job configuration XML logs are created by the Job Tracker. The Job Tracker creates a `.xml` file for every job that runs on the Cluster. These logs are stored in  `${yarn.app.mapreduce.am.staging-dir}`. The `.xml` file describes the job configuration.





## Job Statistics Logs (for Users)

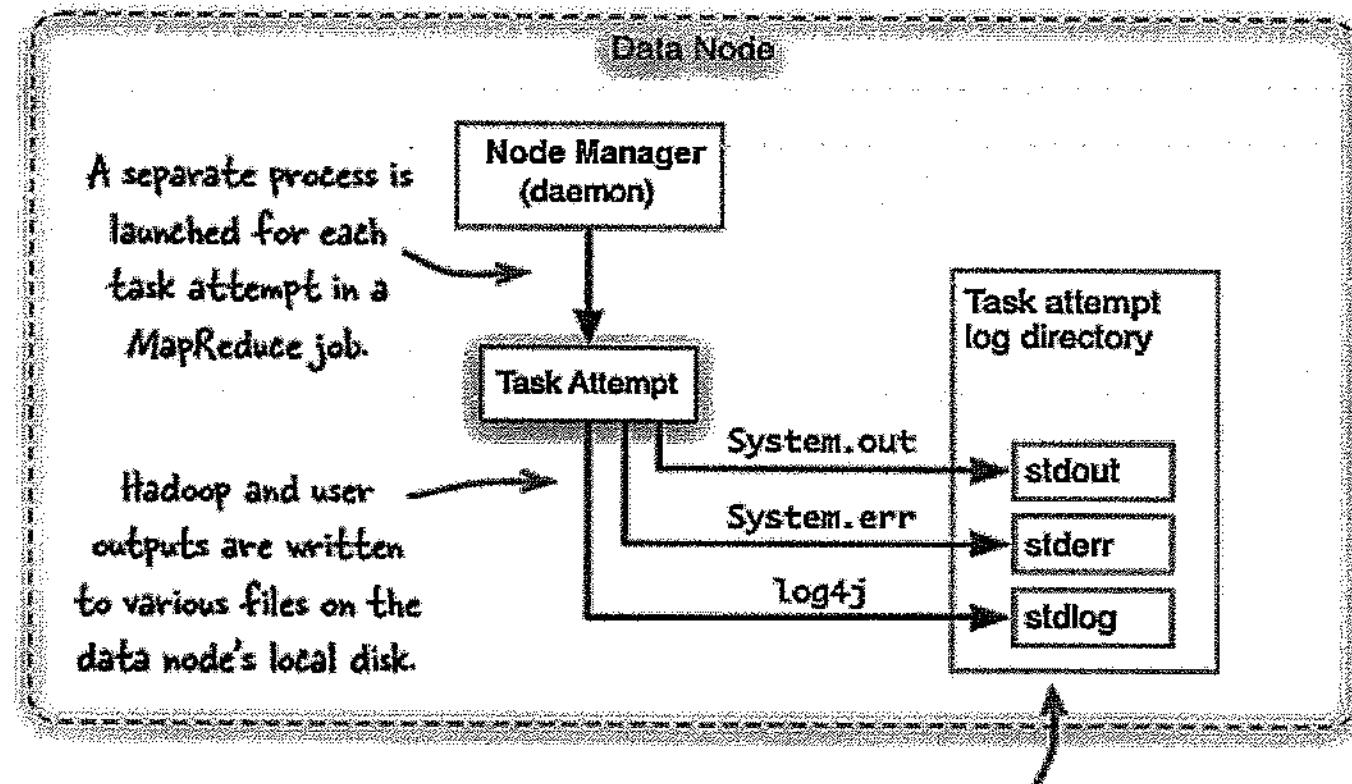
These logs are created by the Job Tracker/Application Master. Runtime statistics from jobs which include task attempts, time spent shuffling, input splits given to task attempts, start times of tasks attempts and other information will be logged into these Logs.

These logs are not rotated. You can clear these logs periodically without affecting Hadoop. However, consider archiving the logs if they are of interest in the job development process. Make sure you do not move or delete a file that is being written to by a running job. Individual statistics logs are created for each job that is submitted to the Cluster. The size of each log file varies. Jobs with more tasks produce larger files.





## Job Logs





## Standard Error Logs (for Users)

These logs are for a particular task attempt and generated by Task Tracker / NodeManager. They contain information written to standard error (*stderr*) captured when a task attempt is run. These logs can be used for debugging. For example, a developer can include *System.err.println("some useful information")* calls in the job code. The output will appear in the standard error files.

These logs are rotated according to the *mapreduce.job.userlog.retain.hours* (default 24 hours) property. You can clear these logs periodically without affecting Hadoop. However, consider archiving the logs if they are of interest in the job development process. Make sure you do not move or delete a file that is being written to by a running job. The size of these log files depends on the amount of *stderr* calls used in the job code.

## Standard Out Logs (for User)

These logs are for a particular task attempt and generated by Task Tracker / NodeManager. These logs are very similar to the standard error logs, except they capture *stdout* instead of *stderr*. File size depends entirely on how much data the task writes to *stdout*.





## System Logs (for Users)

Informational messages from within the task process. These logs contains anything that log4j writes from within the task process. This includes some Hadoop internal diagnostic info. If the job's mapper or reducer implementations include call such as LOG.info(), then that output also get written here. Messages can include information about the task, such as how big its record buffer was or how many reduce tasks there are. The size of these log files depends on the number of log4jcalls used in the job code.

These logs are rotated according to the *mapreduce.job.userlog.retain.hours* (default 24 hours) property. You can clear these logs periodically without affecting Hadoop. However, consider archiving the logs if they are of interest in the job development process.





## Configuring the Logging-Related Parameters in YARN

- *yarn.log.aggregation-enable=true*; If you enable this property, Hadoop collects the logs for each container of an application and moves these files to HDFS once the application is completed.
- *log-aggregation.retain.seconds=604800*; Specifies how long Hadoop will retain the application logs in HDFS.
- *yarn-nodemanager.remote-apps-log-dir=/apps/yarn/logs*; Specifies the directory in HDFS where the application log files are aggregated and stored
- *yarn.nodemanager.log-dirs=\${yarn.log.dir}/userlogs*; Specifies the directories on the Linux file system where YARN sends the application log files. If you enable log aggregation, YARN removes the local files and you can access them through Job History Server.
- *yarn.nodemanager.local-dirs=\${hadoop.tmp.dir}/nm-local-dir*; Specifies the location where YARN stores the intermediate output from MapReduce jobs on the local file system.





## Speculative Execution

- Tasks are not failed, but running slowly or in hung state
- Performance hit to complete the job
- JT / AMa Launches another equivalent & duplicate task as a backup for the problematic task
- Whichever task completed first will be taken into consideration and the other task will be killed automatically
- Controlled by *mapreduce.map.speculative* and *mapreduce.reduce.speculative* properties (by default *TRUE*)



## Failures in Job execution

- Task Failure
  - when user code in map or reduce task throws a runtime exception (*most common*)
  - Sudden exit of the task JVM (may be due to *out of memory* or any other reasons)
  - Hanging tasks, AMa notices that it hasn't received a progress update for a while (*mapreduce.task.timeout* to a default value of 600000 milliseconds)
  - All task failures will be taken care by setting number of maximum attempts (*mapreduce.map.maxattempts / mapreduce.reduce.maxattempts* to a default value of 4)
  - For some applications, it is undesirable to abort the entire job if few tasks fail. Maximum percentage of tasks that are allowed to fail without triggering job failure can be set by using *mapreduce.map.failures.maxpercent* and *mapreduce.reduce.failures.maxpercent*
- Task Killed
  - A task attempt may also be killed, which is different from failing. A task attempt may be killed because it is a *speculative duplicate*
  - If the NodeManager is failed, AMa will mark all the tasks running on that NodeManager as killed.

**RECOVERY PROCESS:** AMa is responsible to reschedule the failed tasks and it will try to not to launch the task on the previously failed NodeManager





## Failures in Job execution (cont'd)

- Application Master Failure
  - AMa sends periodic heartbeats to Resource Manager, and in the event of AMa failure, Resource Manager will detect the failure and start a new instance of AMa running in a new container (managed by a NodeManager)
  - MR AMa will be controlled by *mapreduce.am.max-attempts* property (*by default 2*)
  - YARN AMa will be controlled by *yarn.resourcemanager.am.max-attempts* property (*by default 2*)

**RECOVERY PROCESS:** New AMa will use the job history to recover the state of the tasks that were already run by the failed AMa. Recovery is enabled by default by setting the property *yarn.app.mapreduce.am.job.recovery.enable* to *true*.





## Failures in Job execution (cont'd)

- NodeManager Failure
  - If a NodeManager is failed by crashing, it will stop sending heartbeat to RM. RM will mark the NM as stopped if it hasn't received one for 10 minutes by default.
  - NodeManager failure will be controlled by `yarn.nm.liveness-monitor.expiry-interval-ms` property
  - NodeManagers may be blacklisted if the number of failures (*by default 3*) for a particular application is high, even if the NodeManager itself has not failed.
  - RM does not do blacklisting across all applications, so tasks from new jobs may be scheduled on bad Nodes even if they have been blacklisted by an application master running an earlier job
  - Nodes blacklisting will be controlled by `mapreduce.job.maxtaskfailures.per.tracker` property (*by default 3*)

**RECOVERY PROCESS:** Any task or AMa running on the failed NM will be recovered using the mechanism described in the previous two sections. For the incomplete jobs, AMa will rerun the map tasks that were completed successfully in the failed NM, since their intermediate output residing on the failed NM's local filesystem may not be accessible to the reduce task.





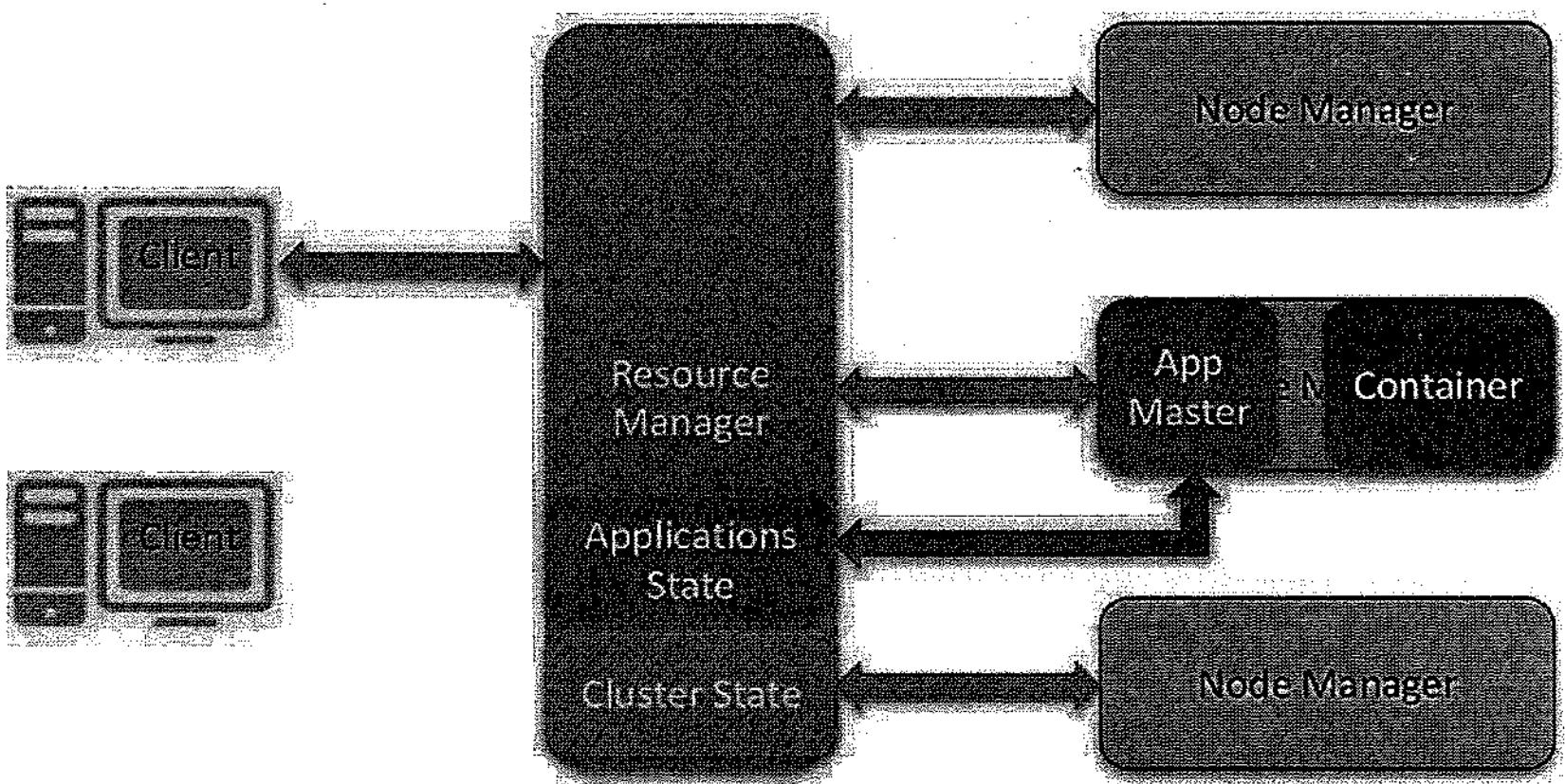
## Failures in Job execution (cont'd)

- Resource Manager Failure
  - Failure of the RM is serious. Without it, neither jobs nor task containers can be launched. In the default configuration, the RM is a single point of failure, since in the (unlikely) event of machine failure, all running jobs will fail, and can't be recovered.
  - To achieve *high availability (HA)*, it is necessary to run a pair of RMs in an active-standby configuration. If the active RM fails, then the standby RM can take over without a significant interruption to the Client.
  - Information about all the running applications is stored in a highly available *Application State Store* (backed by ZooKeeper or HDFS), so that the standby can recover the core state of the failed active RM





## Limitations of Single Resource Manager





## Resource Manager High Availability

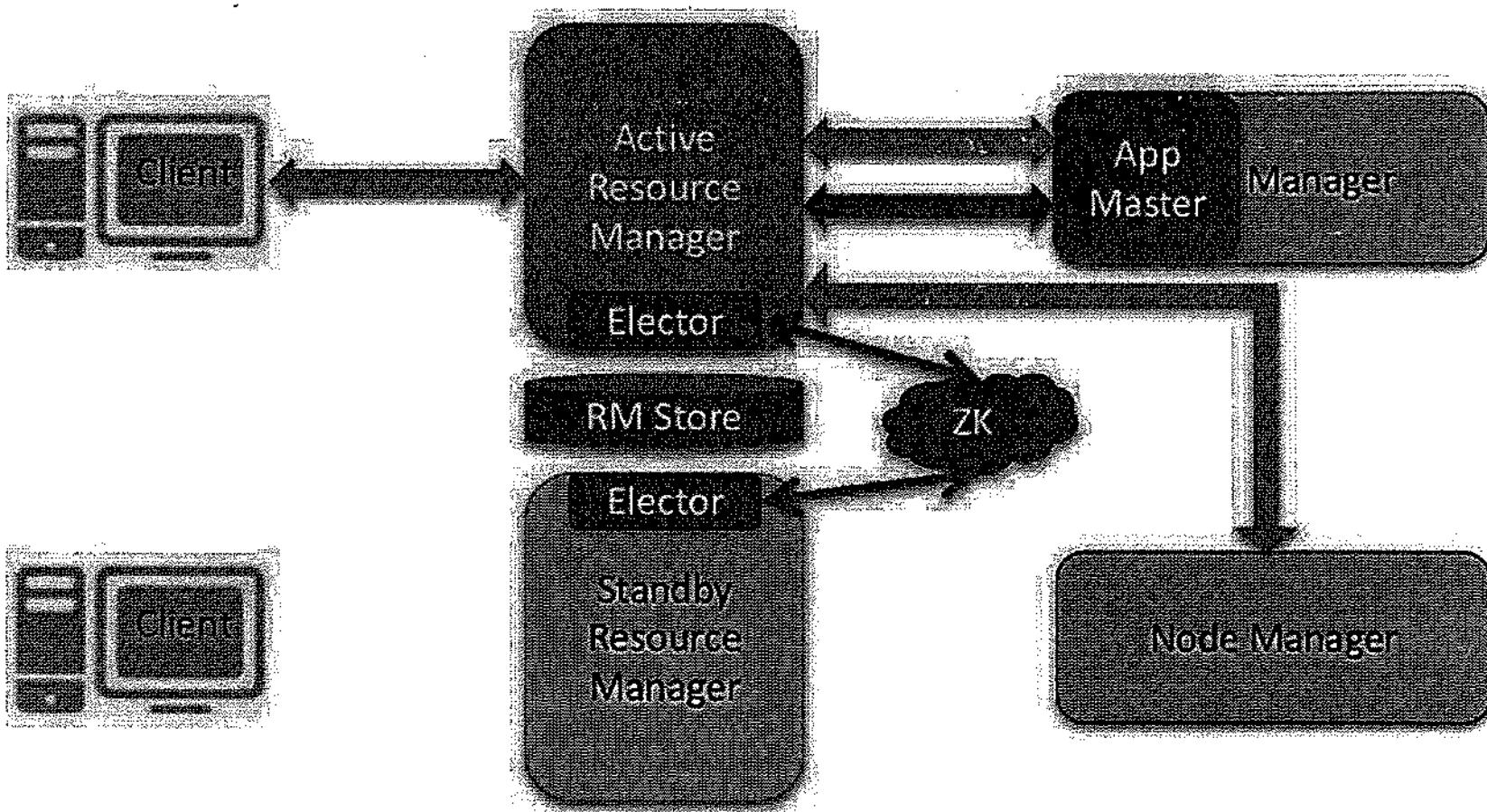
Below are the 2 types of RM Store Implementations:

- **File System Based Store (*FileSystemRMStateStore*)**
  - Any File System: Local, HDFS or any other
- **ZooKeeper Based Store (*ZKRMStateStore*)**
  - Recommended in Production (for Fencing)
  - Loading 10,000 applications takes about 9 seconds





## Resource Manager High Availability (Cont'd)





## Failures in Job execution (cont'd)

- Resource Manager Recovery Process

Resource Manager will hold below information in State Store.

- List of applications that were submitted to RM
- Metadata like job submission context, application-attempt level (AMa) information and credentials about each application
- Overall Cluster state which includes list of Nodes that are active, decommissioned or lost
- Never holds the state of allocated and outstanding containers of each application.

Instead, Resource Manager reconstructs all of the state from the heartbeats of the Nodes and the Application Masters.

After RM restarts, all NodeManagers re-register with new RM and send the information about the containers that are still running as well as those that may have finished. All Application Masters will re-register with RM again and send current state of their corresponding containers. To allocate new containers, RM will wait for a configured amount of time or a certain percentage of previous NodeManagers that have joined with new RM.





## Blacklisting the NodeManagers

When the Application Master submits jobs to the NodeManager and the tasks on that Node have failed too many times, the Application Master will blacklist the NodeManager.

There are two types of NodeManager blacklisting:

- *Per-job blacklisting*
  - AM can blacklist the NM if 3 task failures (`mapreduce.job.maxtaskfailures.per.tracker`) reaches for a particular job
  - Blacklisting option can be enabled by setting `yarn.app.mapreduce.am.job.node-blacklisting.enable` property to `true`
  - AM ignores all blacklisted NMs if 33% threshold reaches
  - AM cannot re-blacklist the same NMs again after ignoring-blacklist event happens once
  - Ignoring-blacklist event only happens once
- *Cluster-wide blacklisting*
  - AM will never launch any new tasks on these Nodes from all jobs
  - Can be used during maintenance activity



## Hadoop File Formats

- Text / CSV / TSV
- JSON (JavaScript Object Notation)
- AVRO
- Sequence
- RC (Record Columnar)
- ORC (Optimized Record Columnar)
- Parquet





## Hadoop File Formats (Cont'd)

### **JSON (*JavaScript Object Notation*)**

- Lightweight data-interchange format
- Language independent
- Self-describing and easy to understand

#### **JSON Example:**

```
{"employees": [  
    {"firstName": "John", "lastName": "Doe"},  
    {"firstName": "Anna", "lastName": "Smith"},  
    {"firstName": "Peter", "lastName": "Jones"}  
]}
```





## Hadoop File Formats (Cont'd)

### *Apache Avro*

Avro stores the data definition in JSON format making it easy to read and interpret; the data itself is stored in binary format making it compact and efficient. Avro files include markers that can be used to splitting large data sets into subsets suitable for MapReduce processing. Avro creates binary structured format that is both compressible and splittable.

### *RC (Record Columnar) File Format*

The RC File is a data storage structure that determines how to minimize the space required for relational data in HDFS. It does this by changing the format of the data using the MapReduce framework. The RC File combines multiple functions such as data storage formatting, data compression, and data access optimization. It is able to meet all the four requirements of data storage

- Fast data storing
- Improved query processing
- Optimized storage space utilization
- Dynamic data access patterns





## Hadoop File Formats (Cont'd)

### *ORC (Optimized Record Columnar) File Format*

The ORC File format provides a more efficient way to store relational data than the RC File, reducing the data storage format by up to 75% of the original. Specifically compared to the RC File, ORC takes less time to access data and takes less space to store data. However, the ORC file increases CPU overhead by increasing the time it takes to decompress the relational data.

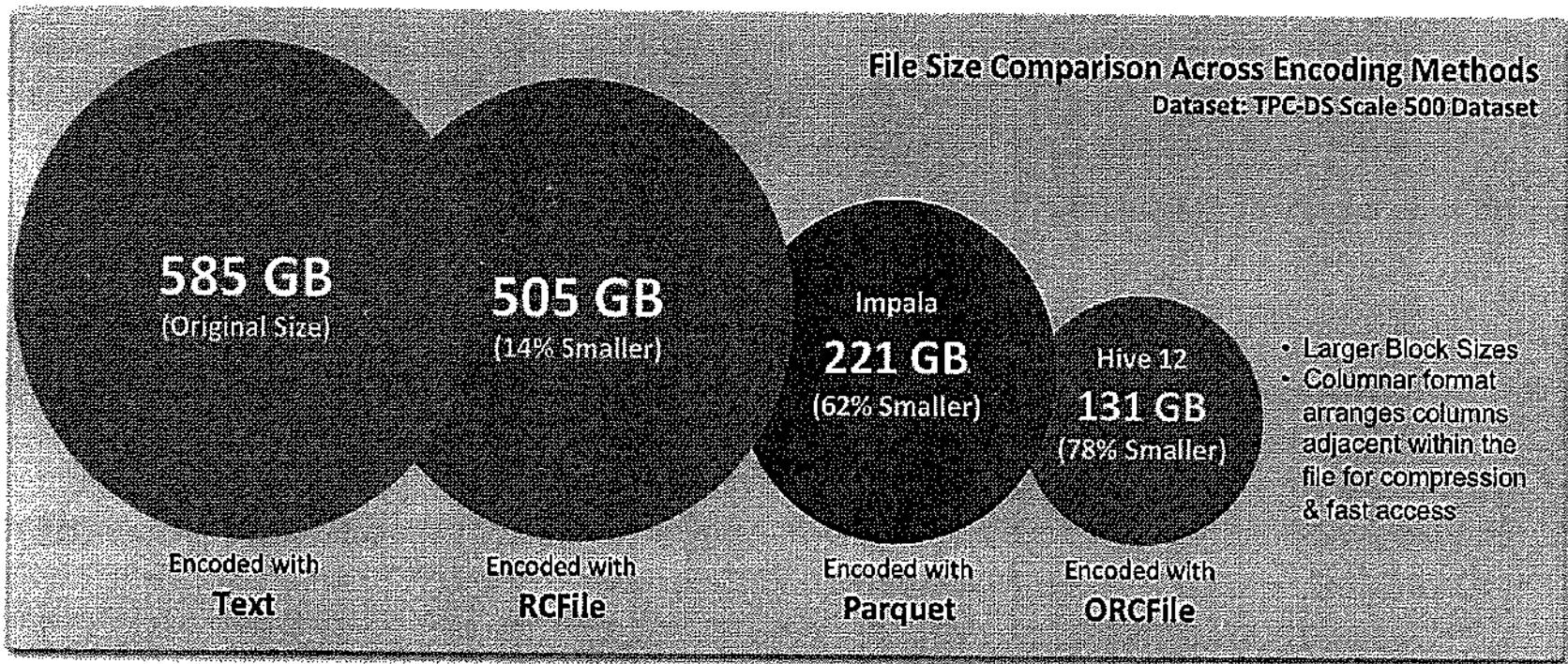
### *Apache Parquet*

Parquet Files are yet another columnar file format like RC and ORC. Parquet enjoys compression and query performance benefits, and is generally slower to write than non-columnar file formats. However, unlike RC and ORC files Parquet SERDEs support limited schema evolution. In Parquet, new columns can be added at the end of the structure. Parquet is supported by Cloudera and optimized for Cloudera Impala.





## Hadoop File Formats (Cont'd)





# Hadoop Distributions





## Hadoop Distributions

**cloudera**  
Ask Bigger Questions

  
**Hortonworks**

  
**MAPR**  
TECHNOLOGIES

**Cloudera** – Purely based on Apache Hadoop with an automated deployment and configuration management tool called “*Cloudera Manager*”

**Hortonworks** – Purely based on Apache Hadoop with an automated deployment and configuration management tool called “*Ambari*”

**MapR** – Has its own proprietary file system (MapR FS), No NameNode concept and an automated deployment and configuration management tool called “*MapR Control System (MCS)*”



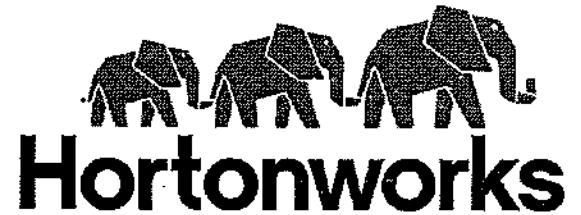
## Different Types of Cloudera Distribution

- Cloudera Express
  - Cloudera Manager Free version
  - Unlimited Nodes for free
- Cloudera Enterprise
  - Cloudera Manager with additional functionality
  - Cloudera Enterprise Data Hub Edition Trial that is valid for 60 days
  - Licensed version





## Different Types of Hortonworks Distributions



	HDP Jump Start	HDP Enterprise	HDP Enterprise Plus
Term	6 Months	1 Year	1 Year
Support Contacts	3	Ranges depending on size of Cluster	Ranges depending on size of Cluster
Support Incidents	Unlimited operational support incidents through development, pilot, staging and deployment	Unlimited operational support incidents through development, pilot, staging and deployment	Unlimited operational support incidents through development, pilot, staging and deployment
Access	Web	Phone & Web	Phone & Web
Hours of Support	Business Hours: Monday - Friday	24 x 7	24 x 7





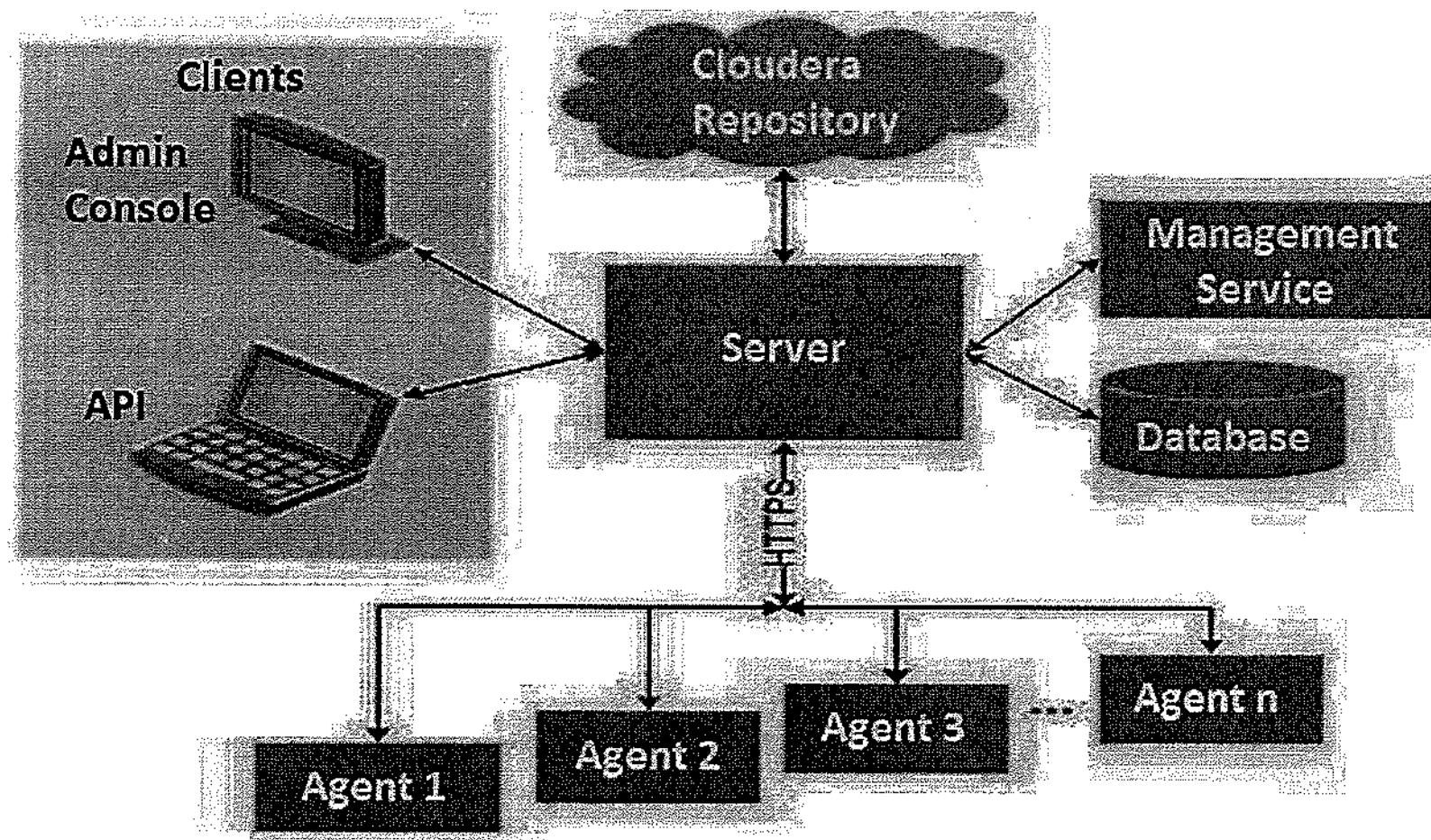
## Different Types of MapR Distributions



- Community Edition
  - For free, unlimited production use with limited features
- Enterprise Edition
  - For critical deployments requiring business continuity (HA/DR)

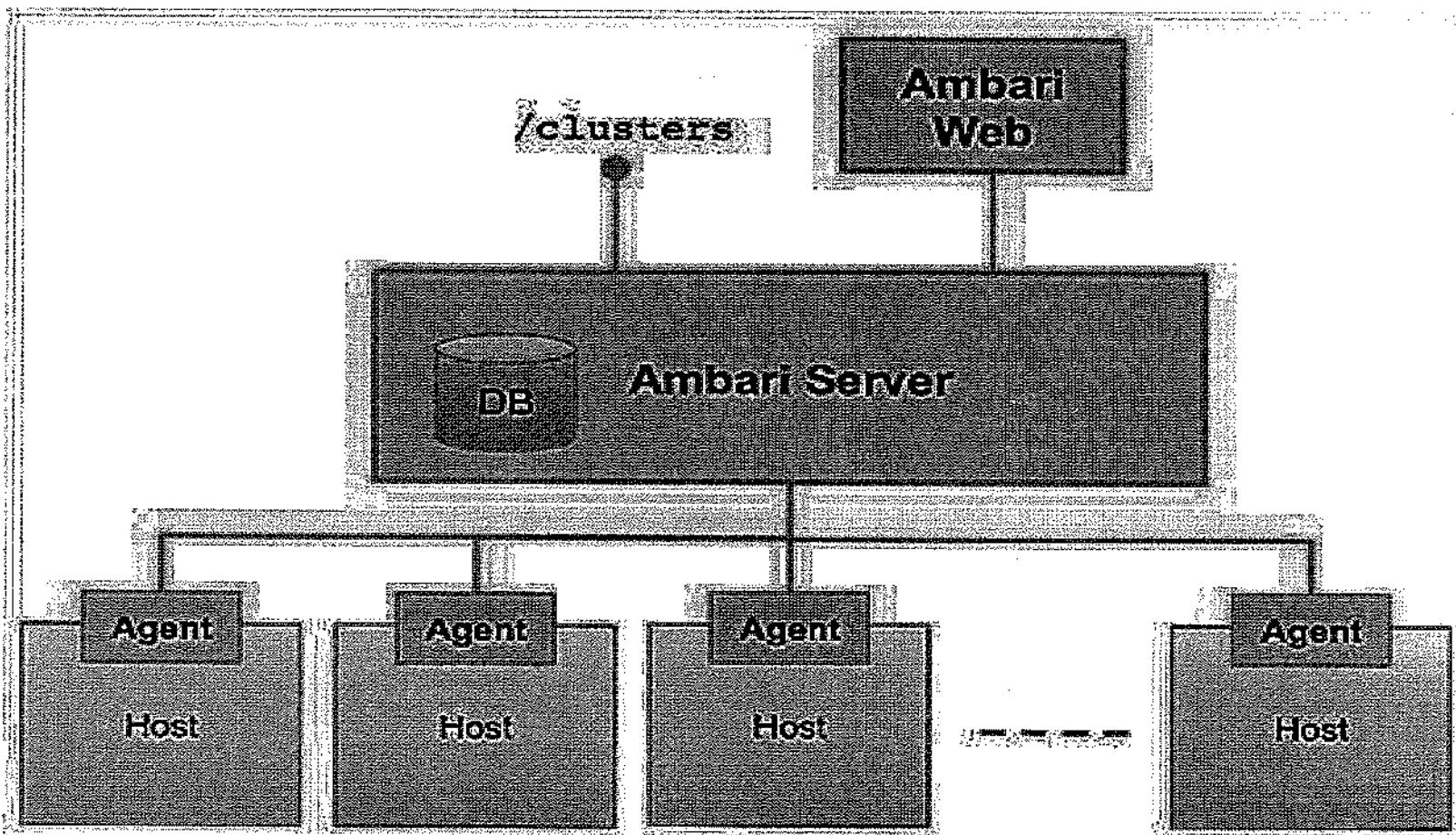


## Cloudera Architecture



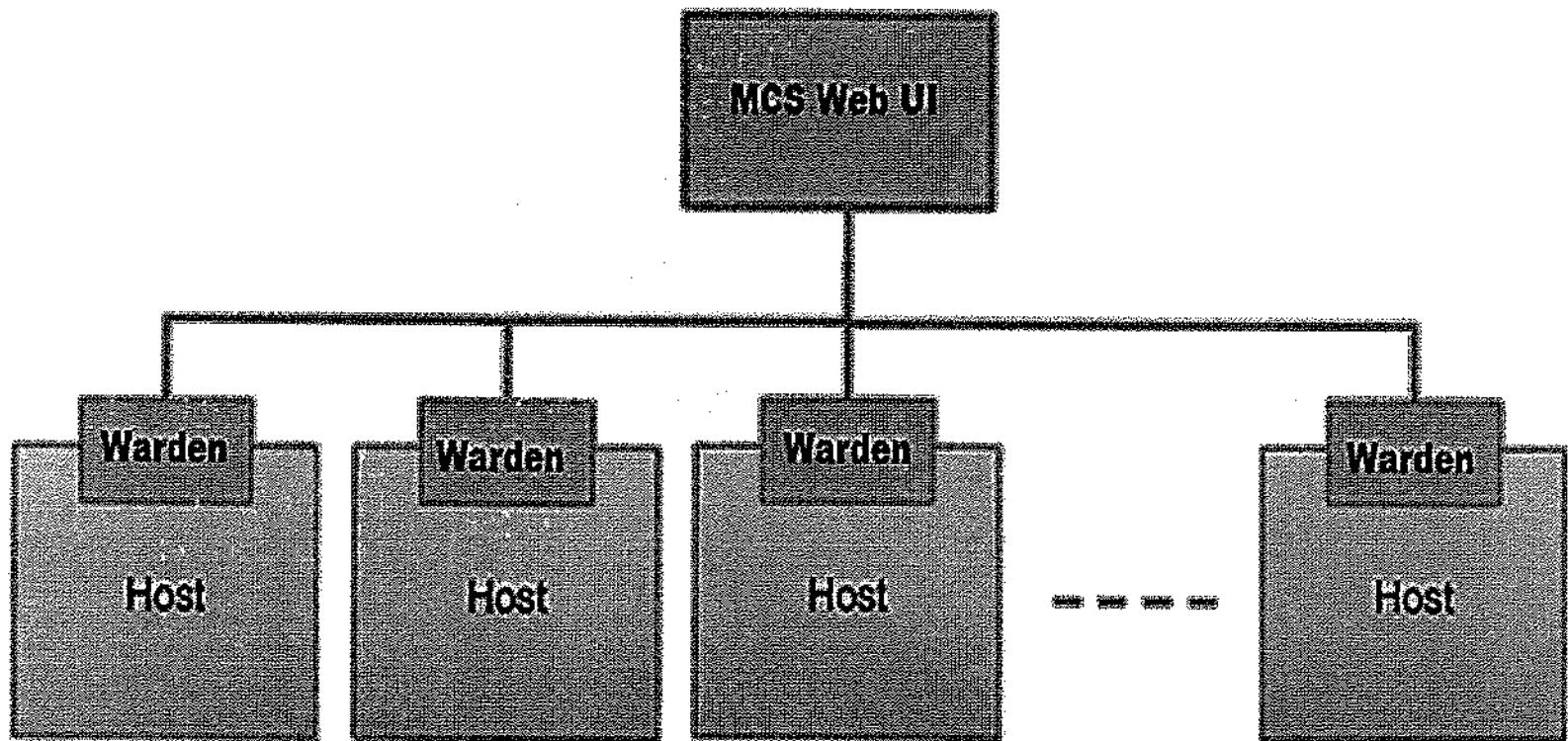


## Ambari Architecture





## MapR Control System (MCS) Architecture





## Cloudera Vs Hortonworks Vs MapR

	Cloudera	Hortonworks	MapR
<b>Performance and Scalability</b>			
Data Ingest	Batch	Batch	Batch & Streaming
Metadata	Centralized	Centralized	Distributed
<b>Dependability</b>			
High Availability	Self-healing across multiple failures	Self-healing across multiple failures	Self-healing across multiple failures
MR High Availability	Continuous without Restart	Continuous without Restart	Continuous without Restart
Upgrading	Rolling Upgrades	Rolling Upgrades	Rolling Upgrades
Replication	Data	Data	Data + Metadata
Snapshots	Consistent only for closed files	Consistent only for closed files	Point-in-time consistency for all files and tables
Disaster Recovery	File Copy Scheduling	File Copy Scheduling	Mirroring
<b>Management</b>			
Management Tools	Cloudera Manager	Ambari	MapR Control System
Volume Support	No	No	Yes



## Cloudera Vs Hortonworks Vs MapR (cont'd)

	Cloudera	Hortonworks	MapR
<b>Heat Map, Alarms, Alerts</b>	Yes	Yes	Yes
<b>Integration with REST APIs</b>	Yes	Yes	Yes
<b>Data and Placement Control</b>	No	No	Yes
<b>Data Access</b>			
<b>File system Access</b>	HDFS, Read-only NFS	HDFS, Read-only NFS	MFS, Read-write NFS
<b>File I/O</b>	Append only	Append only	Read/Write
<b>Security ACLs</b>	Yes	Yes	Yes
<b>Authentication</b>	Kerberos	Kerberos	Kerberos & Native
<b>Authorization</b>	Sentry	Ranger	Native





## Cloudera Vs Hortonworks Vs MapR (cont'd)

	Advantages	Disadvantages
Cloudera (CDH)	<b>CDH has user friendly interface with many features, rich graphics along with some useful tools.</b>	<b>CDH is comparatively slower than MapR Hadoop Distribution.</b>
Hortonworks (HDP)	<b>It is the only Hadoop Distribution that supports Windows platform.</b>	<b>The Ambari Management interface on HDP is just a basic one and does not have many rich features.</b>
MapR	<b>It is one of the fastest Hadoop distribution with multi Node direct access, Snapshots &amp; Mirrors.</b>	<b>MapR don't have a good user interface as Cloudera.</b>





# Hadoop Ecosystem Tools

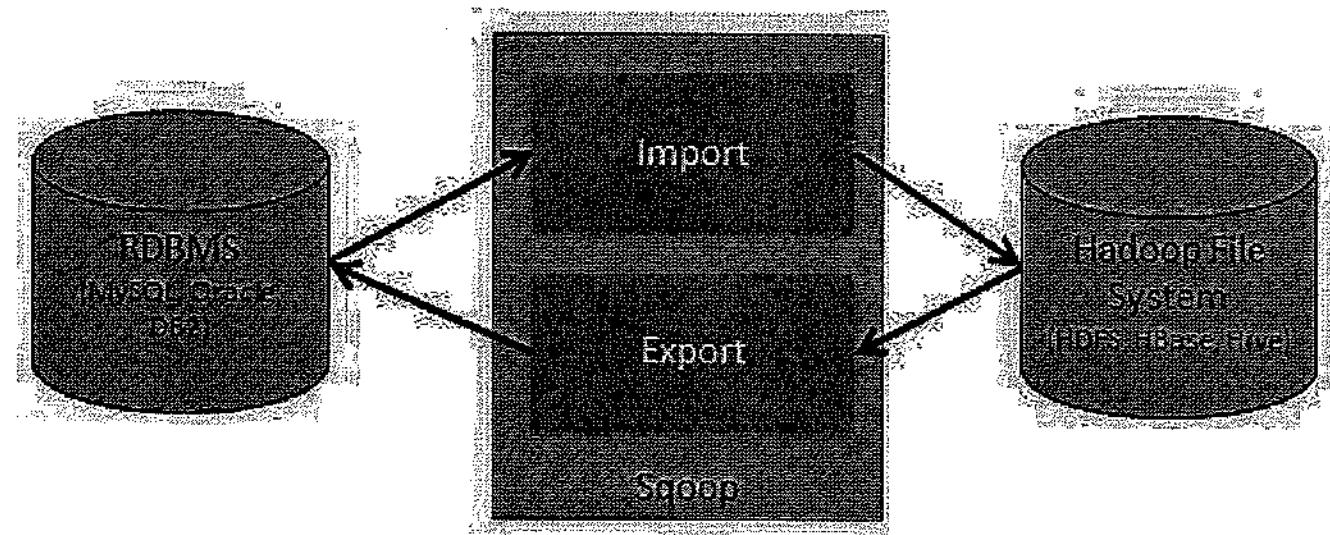


## Sqoop

- Open-source Apache Project
- Originally developed at *Cloudera*
- Designed to import data from RDBMS into HDFS and vice versa
- Supports importing to and exporting from Hive & HBase tables
- Uses JDBC to connect to RDBMS
- Examines each table and automatically generates a Java Class to import data into HDFS
- Creates and runs a *MAP-ONLY MR* jobs to import data
  - By default, 4 Mappers connect to RDBMS
  - Each Mapper imports quarter of the data



## Sqoop Architecture





## Sqoop Features

- Imports a single table, or all tables in a database
- Can specify which rows to import (via a `--where` clause)
- Can specify which columns to import (via a `--columns` clause)
- Can provide an arbitrary `SELECT` statement
- Sqoop can create a Hive table automatically based on the imported data
- Supports incremental imports of data
- Can export data from HDFS to a RDBMS table



## Performance Tuning in SQuoop

Performance mainly depends upon:

- Data Volume
- Source System Performance
- Network Performance
- Hadoop Cluster Performance

In Sqoop:

- Use `-m / --num-mappers` (defaults to 4) option to set number of mappers thereby to increase parallelism in import. Again, too many mappers will increase load on RDBMS.
- Usage of `--direct` in Sqoop command which internally works with `mysqldump`
- Usage of `--split-by` (default is the Primary Key) in case of tables with no index columns
- Usage of `--compress` for data compression  
Ex: `--compress --compression-codec com.hadoop.compression.lzo.LzopCodec`
- Usage of `--batch` while exporting data  
Ex: `sqoop.export.records.per.statement=100 & sqoop.export.statements.per.transaction=100`





## Realtime issues from SQOOP

- Use of incorrect connector for the database
- Missing driver or usage of incorrect driver
- Incorrect approach of providing username/password of the database
- Format of the data stored in HDFS/Hive Tables can create issues. File formats such as ORC files will not allow direct data transfer using Sqoop
- Non-matching or incorrect column names of source and destination tables where HCatalog is used in Sqoop command
- Data type conversion issue
- Delimiter you are using should not be part of data you are importing/exporting
- Table and column names can't have special characters



## Pig

- Open-source Apache Project
- Originally developed at **YAHOO!**
- Client side application for data analysis and processing on Hadoop
- Supports High-level data processing
  - Alternative for writing low-level MapReduce code
  - Especially good at joining and transforming data from files
  - Executes MapReduce jobs
- Provides scripting language known as ***Pig Latin*** for processing data
- PIG interactively use GRUNT shell to execute commands
- Pig Latin script consists of
  - One or more LOAD statements to read in data
  - Transformation statements to process the data
  - DUMP / STORE statements to generate output
- Modes of Running
  - Local Mode              (Single JVM)
  - MapReduce Mode        (Distributed Mode)



## Performance Tuning in PIG

- Usage of data types in the *LOAD* statement
- Try to ignore the unwanted fields to participate in query
- Make your UDFs *Algebraic*
- Drop Nulls Before a Join
- Join Optimizations
- Use the LIMIT Operator
- Prefer DISTINCT over GROUP BY/GENERATE
- Compress the Results of Intermediate Jobs
- Use Specialized Joins
  - *Replicated* Joins
  - *Skewed* Joins
  - *Merge* Joins
- Set number of Mappers
  - *SET pig.maxCombinedSplitSize 250000000;*
- Set number of reducers
  - *SET default\_parallel 100*

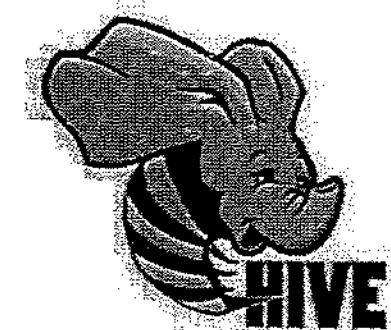


## **Realtime issues from PIG**

- Support a variable number of arguments in UDFs
- UDF support from multiple scripting languages
- Job is not progressing or getting killed
- Map-side joins
- PIG Joins are not used properly based on data type
- Data type conversion issues & Type Casting

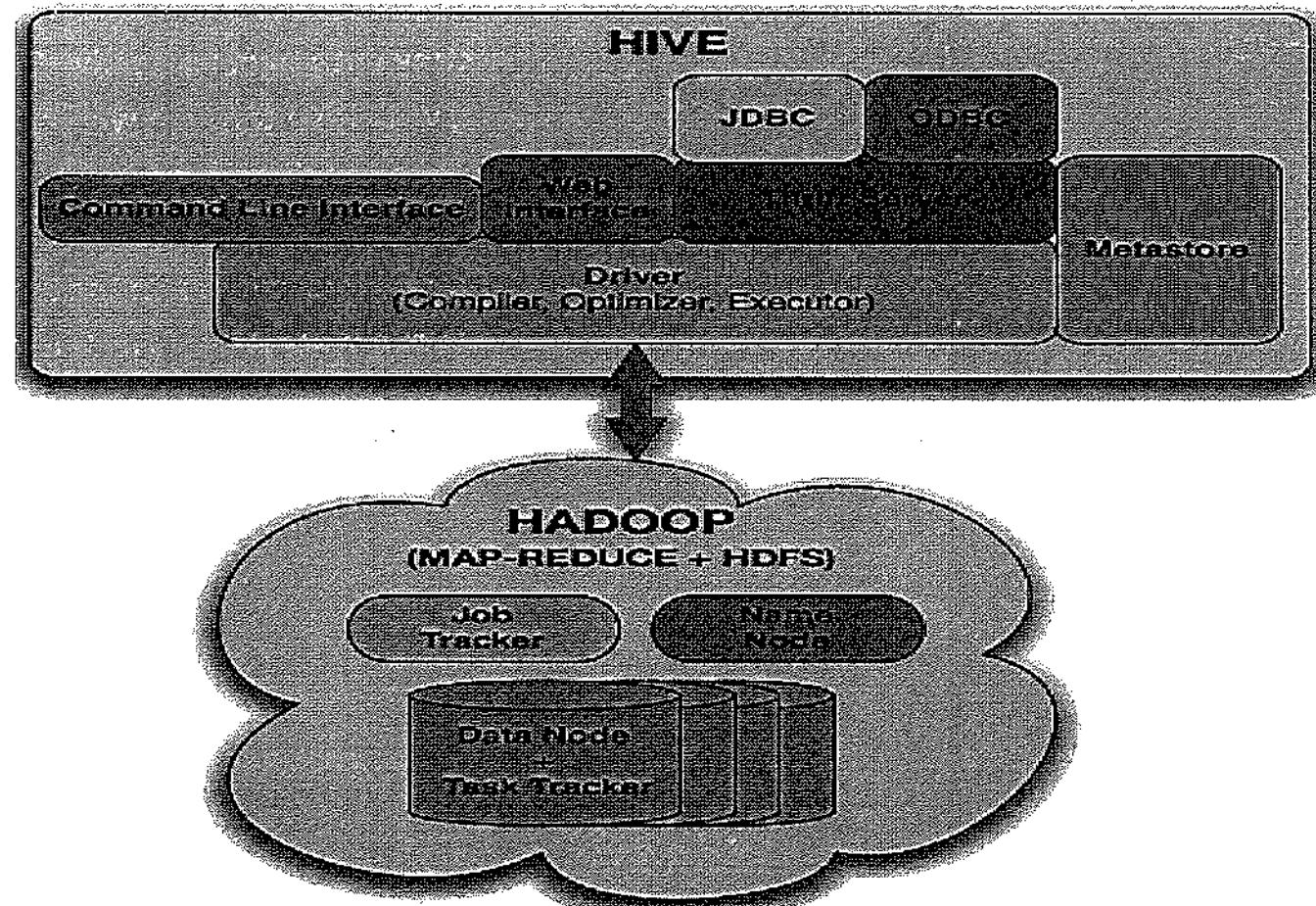
## Hive

- Open-source Apache Project
- Originally developed at *Facebook*
- Data warehousing infrastructure on top of Hadoop
- Supports Hive QL, which is like SQL
- Designed to enable data summarization & ad-hoc querying on very large datasets
- Hive doesn't provide record level insert, update, or delete operations
- Not designed to support Online Transactional Processing (OLTP)





## Hive Architecture





## Hive Architecture (Cont'd)

- **JDBC/ODBC:** Hive clients can directly connect to an underlying metastore using JDBC/ODBC
- **Command Line Interface (CLI):** Most common way of interacting with Hive using Hive Shell
- **Web Interface:** An alternate way to Hive shell for interacting with Hive through web browser
- **Thrift Server:** Hive tools like Beeline and other Web applications can directly connect to Hive and submit their queries.
- **Metastore:** Central repository for Hive metadata. By default, metastore is run in the same process as the Hive service. Using this service, it is possible to run metastore as a standalone process.
- **Driver:** Driver compiles the user queries, optimizes the computation required and executes the required steps with MapReduce jobs.





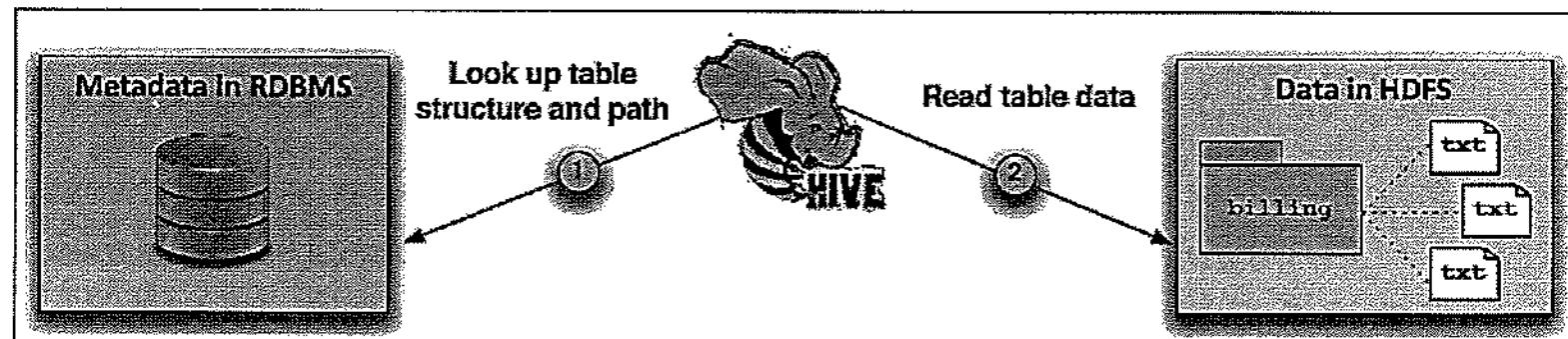
## How HIVE works?

- A Hive table is a directory of data in HDFS with associated metadata
- Tables are created by describing pre-existing data in HDFS

```
CREATE TABLE products (
    id INT,
    name STRING,
    price INT
);

ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

- Metadata (Table Structure & Path to Data) is stored in RDBMS





## Hive Tables

- Hive Table represents a directory on HDFS
  - Default location of a table is */apps/hive/warehouse/<db\_name>.db/<table\_name>*
  - Table can be created by pointing to a specific location (directory) on HDFS
  - Hive interprets all files in the directory as the content of the table
  - Hive stores information about how the rows and columns are delimited within the files in Hive Metastore
- Tables are created as either MANAGED or EXTERNAL
  - MANAGED
    - If the table is dropped, the *schema* and *actual data on HDFS* will be deleted
  - EXTERNAL
    - If the table is dropped, *only the schema* will be deleted, not actual data on HDFS

## Hive Table Partitions

**Hive Table Partitioning** means dividing the table data into parts based on the values of particular columns like Date, Country or City etc., and segregate the input records into different directories based on the selected columns. Partitioning can be done on single column or multiple columns which will impose multi-dimensional structure on directory storage. Partitions can be defined at the time of table creation using **PARTITIONED BY** clause, with a list of columns for partitioning.

### Advantages:

- Partitions can be used to distribute the execution load horizontally
- Query performance will be faster by avoiding full table scan

### Disadvantages:

- Too many partitions will create large number of small files and directories in HDFS, which will increase load on NameNode
- Partitions may not give good performance of some queries on grouping clauses



## Hive Table Bucketing

Tables or partitions are sub-divided into buckets, to provide extra structure to the data that may be used for more efficient querying. Bucketing works based on the value of hash function of some column of a table. Bucketing can be defined at the time of table creation using ***CLUSTERED BY*** clause, with a column for bucketing. Bucketing can be done alone or along with partitions. Bucketed tables will create almost equally distributed data file parts.

### Advantages:

- Since data files are equal sized files, map-side joins will be faster
- Flexibility to keep the records in each bucket to be sorted by one or more columns

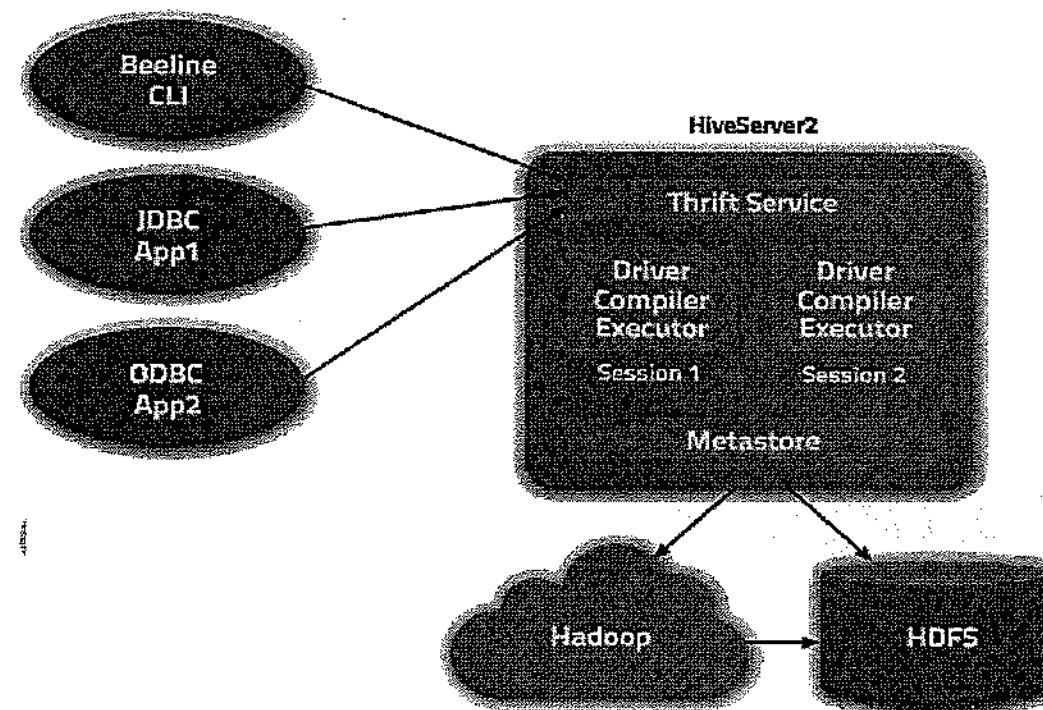
### Disadvantages:

- Too many small buckets will increase load on NameNode
- Specifying bucketing doesn't ensure that the table is properly populated. Data loading into buckets needs to be handled by our-self.



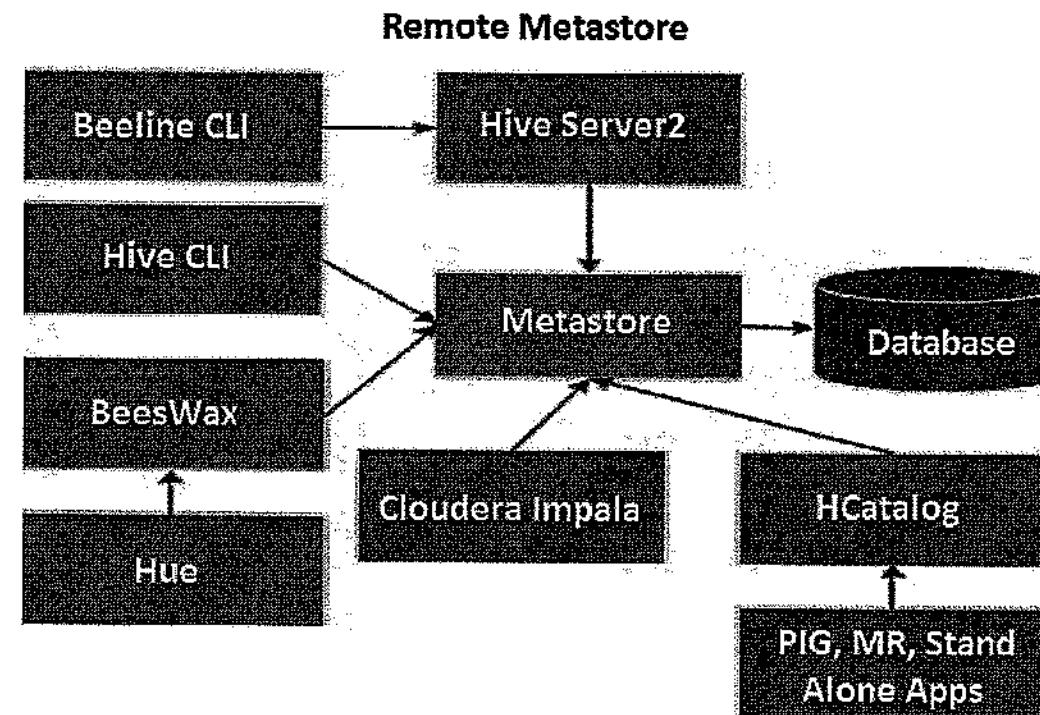
## Hiveserver2 (HS2)

HiveServer2 (HS2) is a server interface that enables remote clients to execute queries against Hive and retrieve the results. The current implementation, based on Thrift RPC supports multi-client concurrency and authentication. Hive clients (like *Beeline*, *Squirrel*) and external Web applications can connect to HiveServer2 through Thrift and submit their queries.



## Hive Metastore

Hive Metastore is a central repository for Hive metadata. By default, metastore is run in the same process as the Hive service. Using this service, it is possible to run metastore as a standalone process to serve all its clients.





## Hive Table Locks

The following lock modes will be defined in Hive.

- Shared Lock – for any kind of select queries
- Exclusive Lock – for any kind of insert, drop and few alter table cases

To debug hive locks:

**SHOW LOCKS <TABLE\_NAME>;**

**SHOW LOCKS <TABLE\_NAME> EXTENDED;**

**SHOW LOCKS <TABLE\_NAME> PARTITION (<PARTITION\_DESC>) EXTENDED;**

**LOCK TABLE TABLE\_NAME SHARED;**

**UNLOCK TABLE IF LOCK EXISTS TABLE\_NAME;**

*hive.lock.manager* maintains all table locks within zookeeper memory. This is the new feature in Hive to manage locks. If a heartbeat is not received in the configured amount of time, the lock will be aborted automatically.



## Security on HIVE

- *Storage Based Authorization in the Metastore Server*
  - Hive uses read/writes permissions for directories on HDFS. When any Clients tries to access any Databases, Tables and Partitions, it checks if the Clients are having permissions on corresponding directories on HDFS. With the help of HDFS ACLs, we can have a lot of flexibility in controlling access to HDFS.
- *SQL Standards Based Authorization in HiveServer2*
  - This is to control the Hive authorization at finer levels such as columns and views. It is based on the SQL standards for authorization, and uses the familiar grant/revoke statements to control access. It needs to be enabled through HiveServer2 configuration. SQL Standards Based Authorization will be disabled when you are using Hive through Command Line Interface.
  - Privileges, Users and Roles can be used to grant/revoke specific permissions on Hive.
- *Default Hive Authorization (Legacy Mode)*
  - Bit similar to RDBMS style authorization model but any user can grant/revoke permissions to himself. Authorization can be set at different levels like Users, Groups and Roles.





## Security on HIVE with Apache Sentry

Apache Sentry is a granular, role-based authorization module for Hadoop. Sentry provides the ability to control and enforce precise levels of privileges on data for authenticated users and applications on a Hadoop cluster. Sentry currently works out of the box with Hive, HCatalog, Solr, Impala.

Ex:

```
CREATE ROLE admin_role;
```

```
GRANT ALL ON SERVER server1 TO ROLE admin_role;
```

```
GRANT ROLE admin_role TO GROUP hdpadmin;
```

```
CREATE ROLE read_retail_role;
```

```
GRANT SELECT ON DATABASE retail TO ROLE read_retail_role;
```

```
GRANT read_retail_role TO group blrproj;
```

```
show roles;
```



## Where / When to use HIVE?

- Hive should be used to store and process structured data
- Hive should be used for analytical querying of data collected over a period of time. For example, to know the sales trend for last 5 years
- Data Summarization & Ad-hoc querying
- When we don't have any updates on data (usually OLAP)
- Hive is not meant for Real-time Reporting purpose (OLTP)



## Performance Tuning in Hive

- Use Partitions and Buckets
- Enable Optimization

```
set hive.auto.convert.join = true  
set hive.optimize.skewjoin = true
```

- Controlling split size:

```
set hive.input.format=org.apache.hadoop.hive ql.io.HiveInputFormat;  
set mapreduce.input.fileinputformat.split.minsize=100000000;  
set mapreduce.input.fileinputformat.split.maxsize=300000000;
```

- Set number of mappers by

```
set mapreduce.job.maps = 20;  
set mapreduce.job.reduces = 20;
```

- Use Apache Tez execution engine instead of Map-reduce engine

```
set hive.execution.engine=tez;
```

- Use ORC file format

- ORC supports compressed storage (with **ZLIB** or with **SNAPPY**) but also uncompressed storage.



## Performance Tuning in Hive (Cont'd)

- **Use Vectorization**
  - Vectorized query execution improves performance of operations like scans, aggregations, filters and joins, by performing them in batches of 1024 rows at once instead of single row each time

```
set hive.vectorized.execution.enabled = true;
set hive.vectorized.execution.reduce.enabled = true;
```
- **Cost Based Query Optimization**
  - Cost-based optimization, performs further optimizations based on query cost, resulting in potentially different decisions: how to order joins, which type of join to perform, degree of parallelism and others.

```
set hive.cbo.enable=true;
set hive.compute.query.using.stats=true;
set hive.stats.fetch.column.stats=true;
set hive.stats.fetch.partition.stats=true;
```
  - Prepare the data for CBO by running Hive's "analyze" command to collect various statistics on the tables for which we want to use CBO.

```
analyze table tweets compute statistics;
analyze table tweets compute statistics for columns sender, topic;
```





## Realtime issues from Hive

- Hive Services going down with out of memory exceptions
- HiveServer2 is down
- Hive Metastore Service is not responding
- Credential issues when connecting from external web applications
- Support of correlated subqueries in the WHERE clause
- Data type conversion issues & Type Casting
- Hive Locking issues
- Beeline connectivity
- SerDe issues
- Hive queries over HBase tables
- Parallel execution of independent operators in a plan





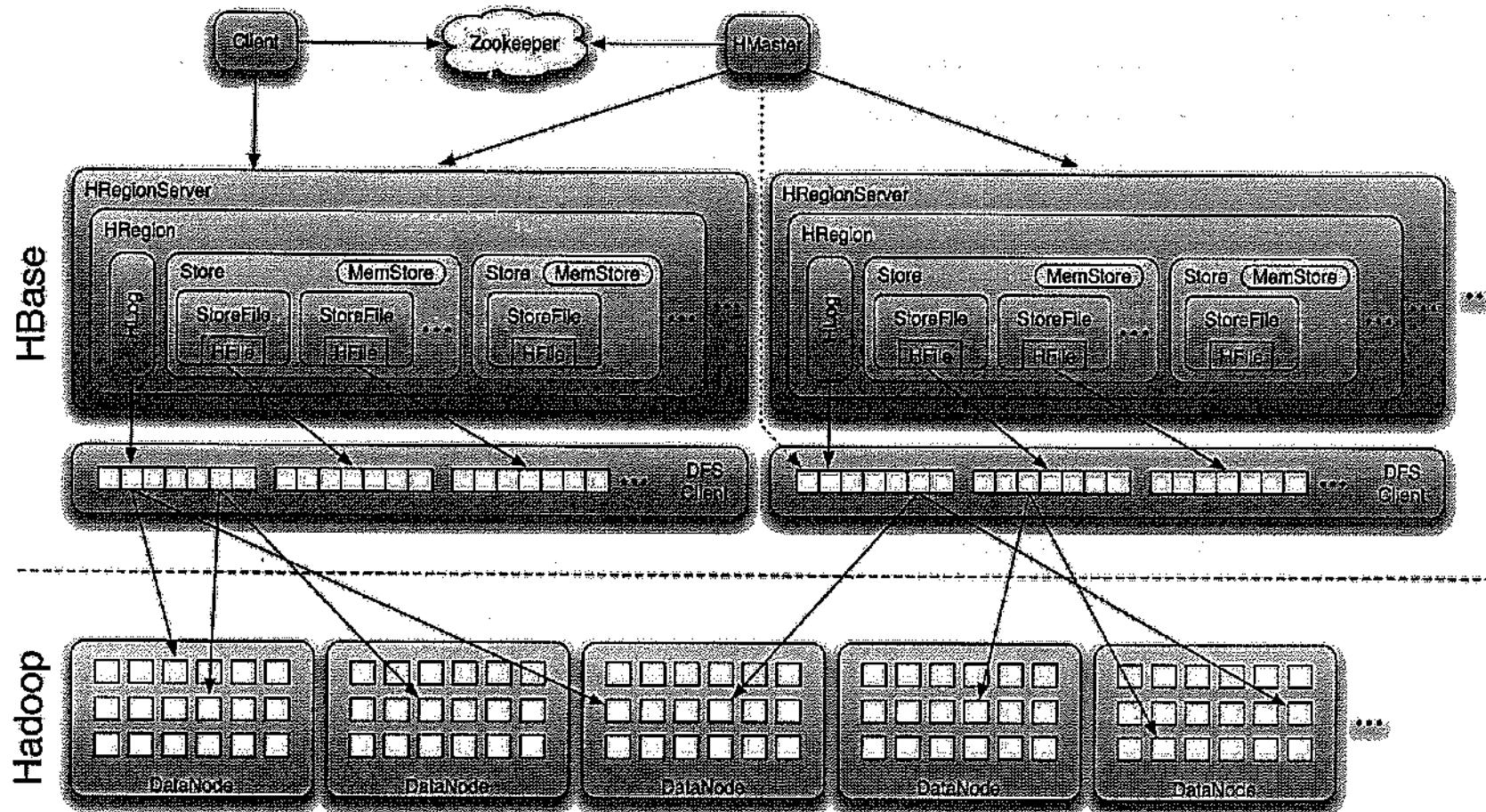
## HBase

- Open-source Apache Project
- Modeled after *Google's BigTable*
- Column Oriented Database on top of Hadoop
- Read / Write access to data on HDFS
- Multi-Dimensional (*Versions*)
- Dynamic Schema
- Automatic Partitioning / Auto Indexing
- Not a SQL Database – No Joins, No Query Engine, No Data types, No SQL, No Schema
- HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups





## HBase Architecture



## HBase Components

- **ZooKeeper**
  - Provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
  - ZooKeeper has Z-Nodes representing different region servers. Master servers use these Nodes to discover available servers and to track server failures.
  - Clients communicate with region servers via ZooKeeper
  - In pseudo and standalone modes, HBase itself will take care of ZooKeeper (*Internal*)
- **HMMaster**
  - Responsible for schema changes and other metadata (*META*) operations such as creation of tables and column families
  - Assigns regions to the region servers and takes the help of ZooKeeper for this task
  - Handles load balancing of the regions across region servers
  - Unloads the busy Regions Servers and shifts the regions to less occupied Region servers

## HBase Components (cont'd)

- **HRegion Server**

The Region Servers have regions that

- Communicate with the Client and handle data-related operations
- Handle read and write requests for all the regions under it
- Decide the size of the region by following the region size thresholds

- **Regions**

- Basic element of availability and distribution for tables, and are comprised of a Store per Column Family.

- **Store**

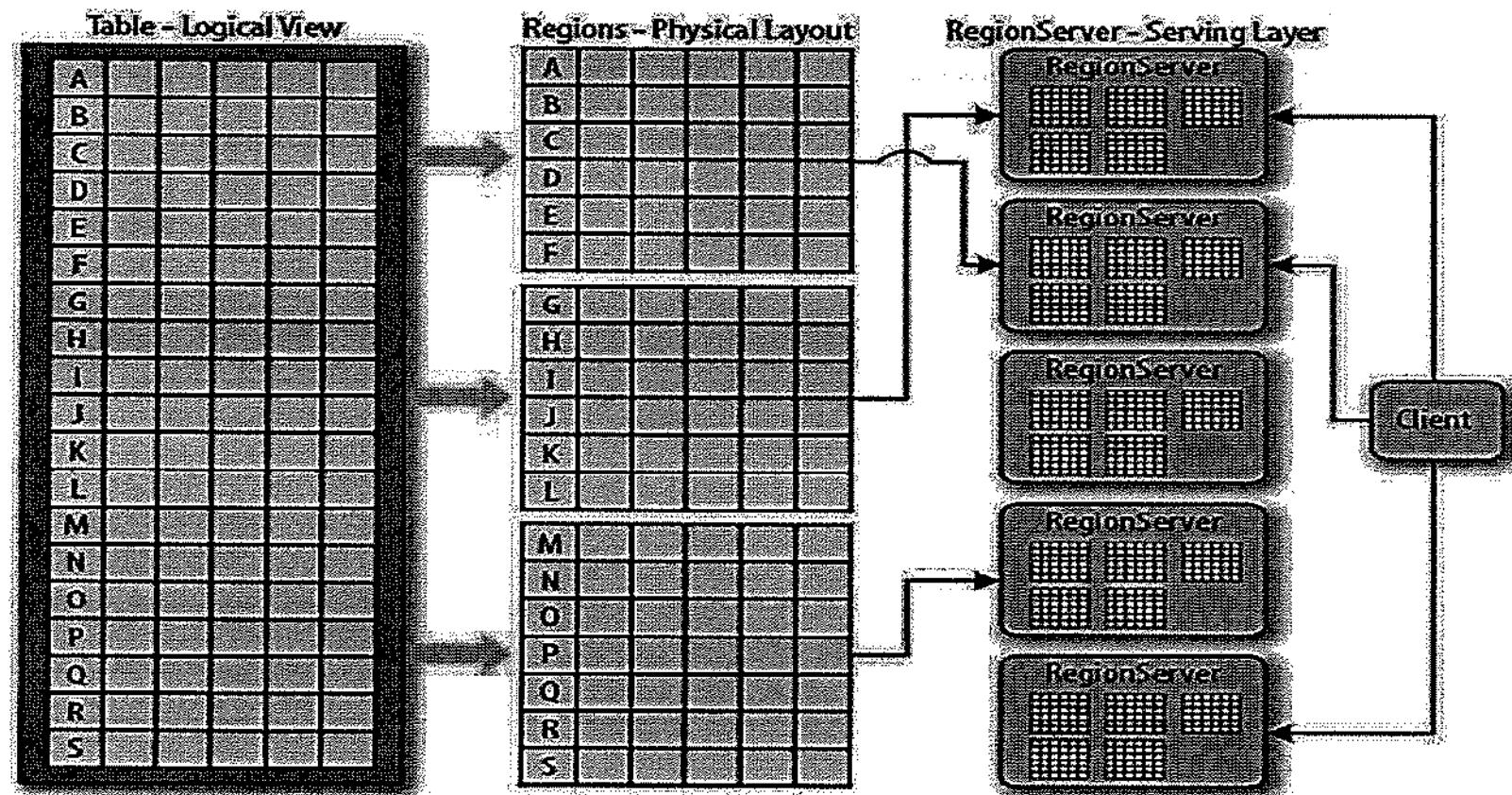
- It hosts a MemStore and 0 or more StoreFiles (*HFiles*). A Store corresponds to a column family for a table for a given region.



## HBase Components (cont'd)

- **MemStore**
  - Each change is stored in a place in memory called memStore, which efficiently supports random writes. Data in the memStore is stored in the same manner as data in HFile. When memStore accumulates enough data, the entire sorted set is written to a new HFile.
- **StoreFile (HFile)**
  - This is the place where your data lives.
- **WAL / HLog**
  - This is the place where all logs are written about each record.

## HBase Auto Sharding





## HBase Performance Tuning

- Less number of column families
- Enable Block Cache
- Usage of Compression Techniques
- Usage of Compaction Scripts
- Remove deleted or expired cells
- Need to limit number of HFiles (fewer larger ones)





## Realtime issues from HBase

- HBase Services going down with out of memory exceptions
- HBase Thrift is down
- Unable to connect to HBase from external web applications
- HRegions are going offline
- Closing and Assigning/Unassign the HRegions
- Problems during bulk load process
- Join issues when integrated with Hive
- Major & Minor Compactions



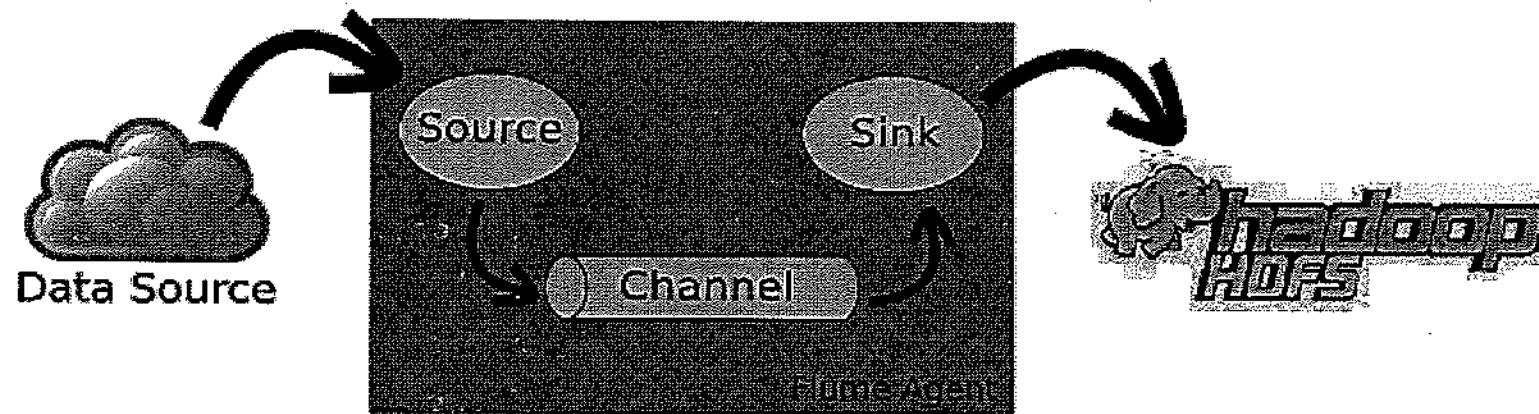


## Flume

- Open-source Apache Project
- Initially developed by *Cloudera*
- Distributed, reliable, available service for efficiently moving large amount of data as it is produced
  - Ideally suited for gathering logs from multiple systems and inserting them into HDFS *as they are generated*
- Typically used to ingest log files from real-time systems such as web servers, firewalls, and sensor networks into HDFS
- Currently in use in many large Organizations, ingesting millions of events per day



## Flume Architecture



**Each Flume Agent has a Source and Sink**

- **Source**
  - Tells the Node where to receive data from
- **Sink**
  - Tells the Node where to send data to



## Flume Architecture (cont'd)

- Channel
  - A queue between the Source and Sink
  - Can be in-memory only or '*Durable (Disk based)*'
  - Memory channels will lose data if power is lost
  - Durable (Disk based) channels guarantees durability of data in case of a power loss
- Data Transfer between Agents and Channels is transactional
  - A failed data transfer to a downstream agent rolls back and retries
- Can configure multiple agents with the same task (*High Availability*)
  - E.g., two Agents doing the job of one 'collector'. If one agent fails, then upstream agents would fail over
- Scalability
  - The ability to increase system performance linearly or better by adding more resources to the system
  - Flume scales horizontally
    - As load increases, more machines can be added to the configuration

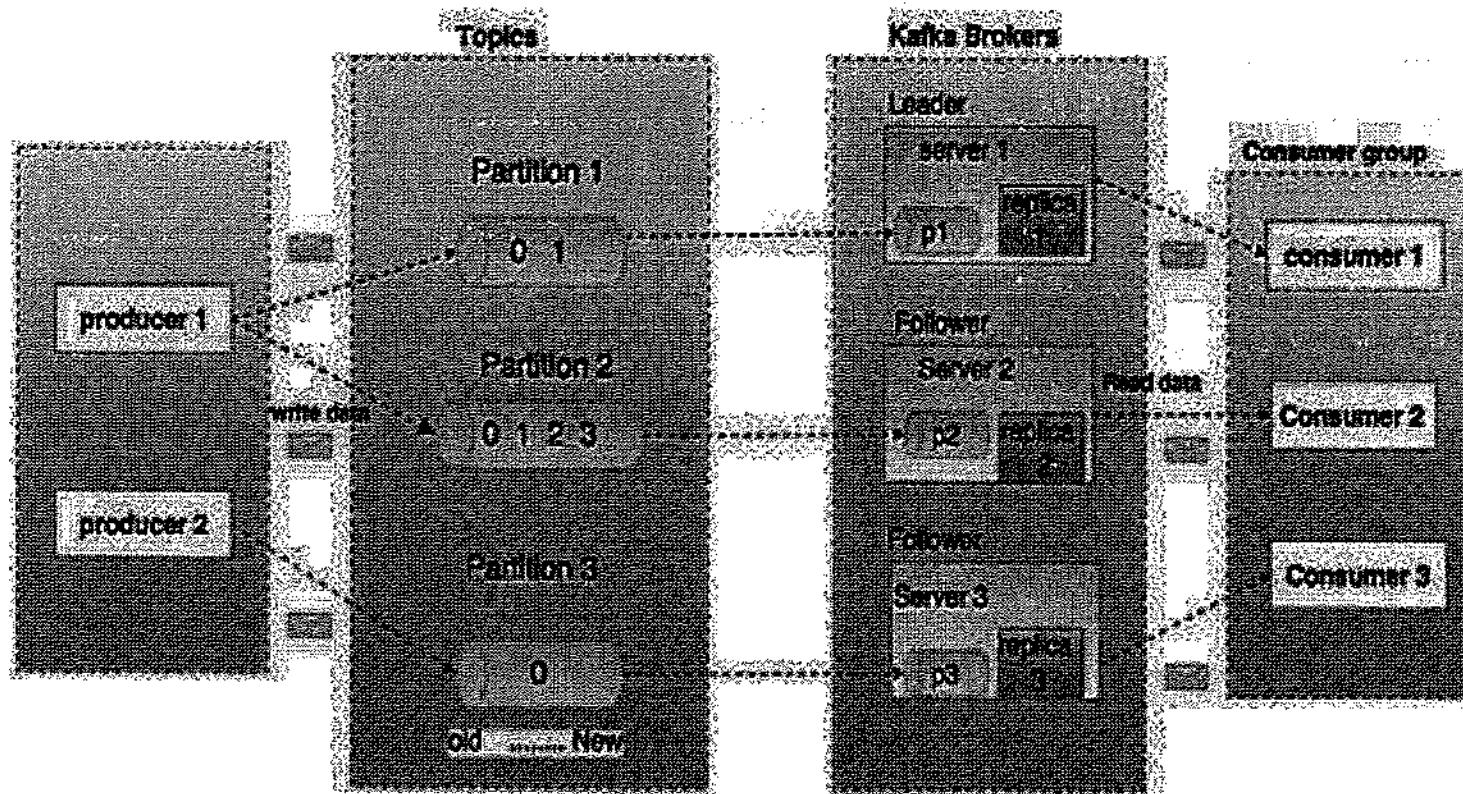


## Kafka

- Open-source Apache Project
- Initially developed by *LinkedIn*
- Better throughput, built-in partitioning, replication and fault-tolerance for large-scale message processing applications
- Widely used distributed Publish-Subscribe system, where message producers are called publishers and message consumers are called subscribers
- Suitable for both offline and online message consumption where Kafka messages are persisted on the disk and replicated within the cluster to prevent data loss
- Can be integrated very well with Storm and Spark for real-time streaming data analysis



## Kafka Architecture



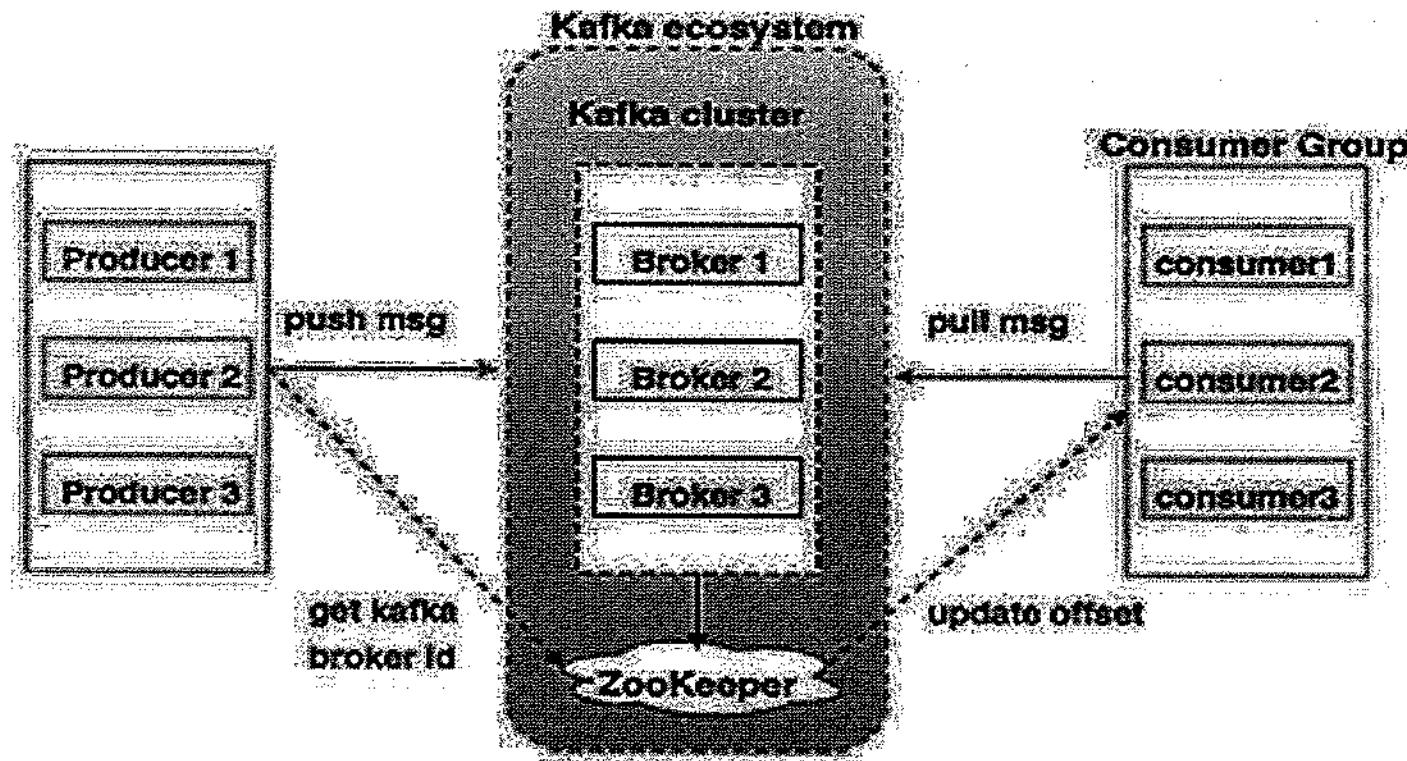
## Kafka Components

- **Kafka Cluster:** A system with a combination of one or more Brokers
- **Broker:** Brokers can handle zero or more partitions per topic
- **Topics:** Stream of messages from a particular category
- **Partitions:** Topics are divided into many partitions
- **Partition Replicas:** Backup copy of partitions
- **Leader Partition:** Leader is the node responsible for all reads and writes for a given partition
- **Follower Partition:** Follower is standby for a Leader. If Leader fails, one of the follower will automatically become Leader
- **Producer:** Publisher of messages to one or more topics
- **Consumer:** Consumers read data from topics and consumes published messages by pulling data from brokers
- **Zookeeper:** To manage and coordinate Kafka brokers





## Kafka workflow



## Kafka workflow (Cont'd)

- Producers send messages to a topic at regular intervals based on size or duration
- Kafka broker stores all messages in the partitions for that particular topic and ensures the messages are equally shared between partitions
- Consumer subscribes to a specific topic
- Kafka provides the current offset of the topic to the Consumer on frequent intervals
- Once Kafka receives the messages from Producer, it forwards these messages to the Consumers
- Consumer receives the messages and process them
- Once the messages are processed, consumer will send an acknowledgement to the Kafka broker
- Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper. Since offsets are maintained in the Zookeeper, the consumer can read next message correctly even during server outages.
- This above flow will repeat until the consumer stops the request
- Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages





## Realtime issues from Kafka

- Kafka Services going down with out of memory exceptions
- Session Timeouts due to Zookeeper issues
- Due to mismatch of Topic Offset values, consumers are unable to get the data
- Insync Replicas are failing due to message size of the topic

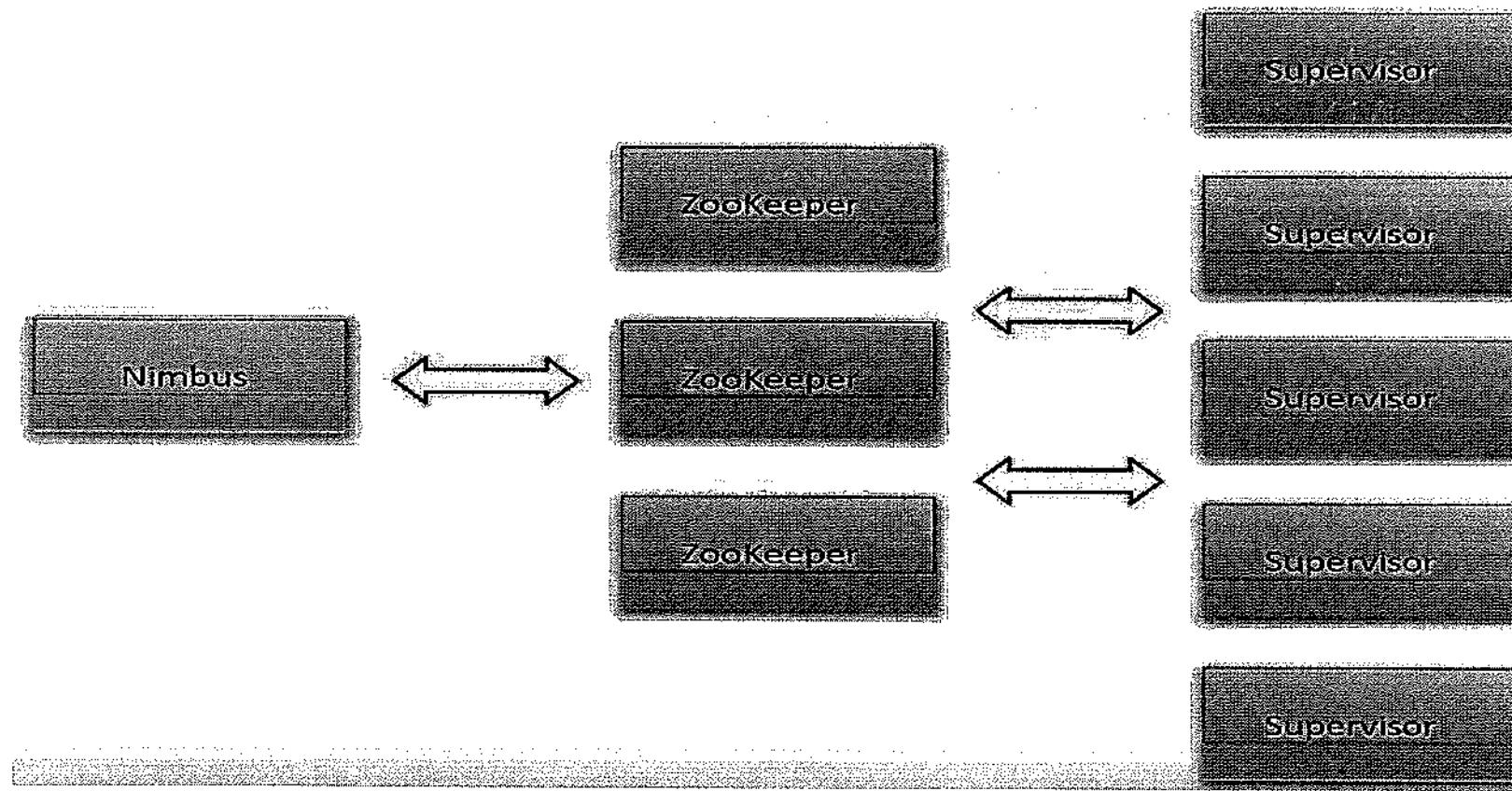
## Storm

- Open-source Apache Project
- Initially developed by *Nathan Marz* and acquired by *Twitter*
- Reliable, distributed and fault tolerant real-time processing system, similar to Hadoop
- Mainly used to process batch data, real-time data and real-time analytics
- The input stream of a Storm cluster is handled by a component called *Spout*
- The Spout passes the data to a component called a *Bolt*, which transforms it in some way
- A Bolt either persists the data in some storage or passes to other some other Bolt





## Storm Architecture



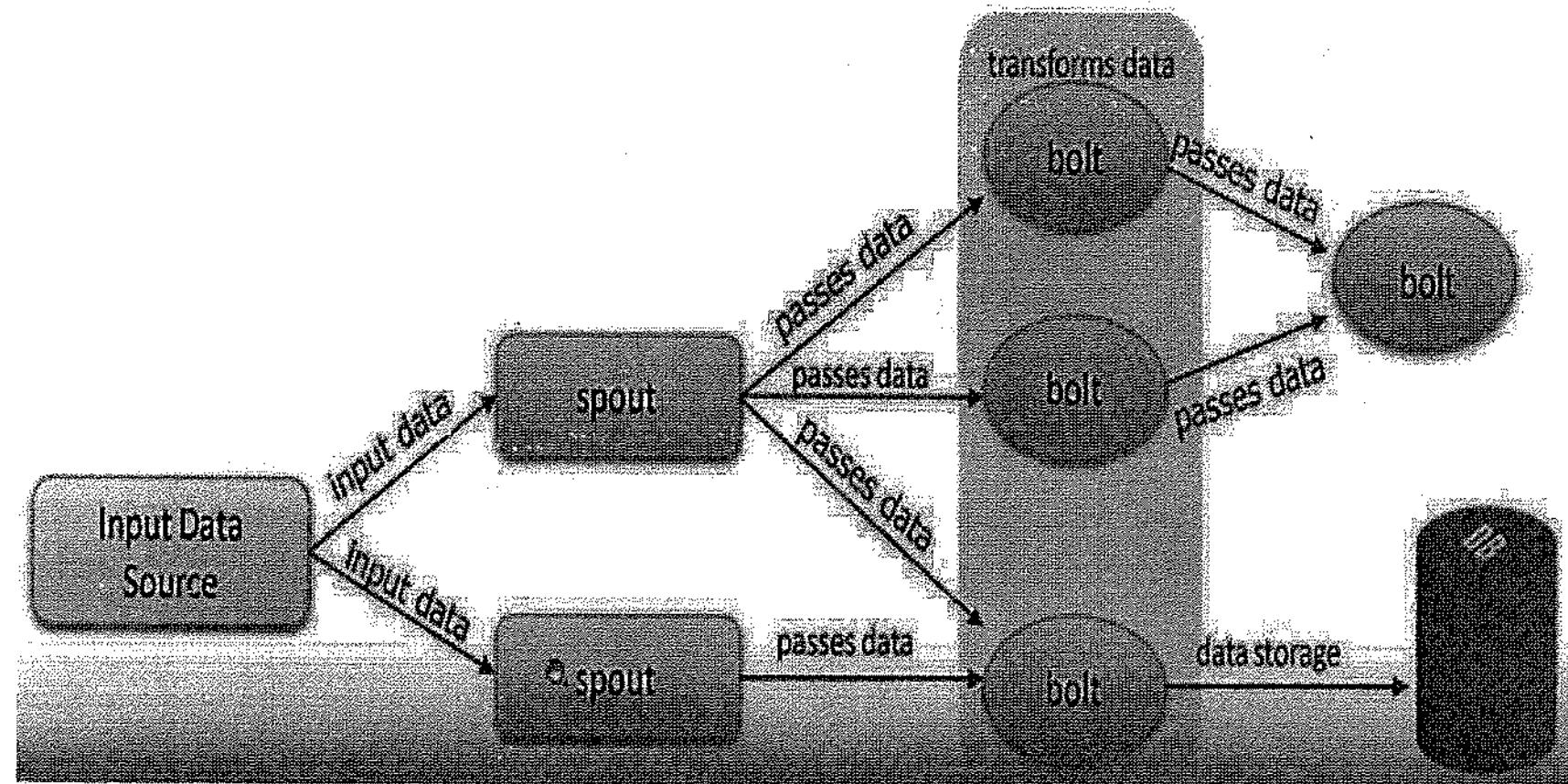
## Storm Components

- **Nimbus (Master Node, Similar to RM)**
  - Uploads computations for computation
  - Distributes the code across the Storm Cluster
  - Launches workers across the Storm Cluster
  - Monitors computation and reallocates the workers as needed
- **Zookeeper**
  - Coordinates the Storm Cluster
- **Supervisor Nodes**
  - Communicates with Nimbus through Zookeeper, starts and stops workers according to signals from Nimbus

## Storm Components (Cont'd)

- *Tuples*
  - An ordered list of elements
- *Streams*
  - An unbounded sequence of tuples
- *Spouts*
  - Sources of Streams in a computation
- *Bolts*
  - Process input streams and produces output streams. They can run functions, filter and aggregate the data or talk to databases to store the data
- *Topologies*
  - The overall calculation, represented visually as a network of Spouts and Bolts

## Storm Process Flow





## Spark

- Open-source Apache Project
- Lightning-fast Cluster computing technology on top of Hadoop
- Designed for fast computation with the help of in-memory computing
- Designed to run batch jobs, iterative algorithms, interactive queries and streaming



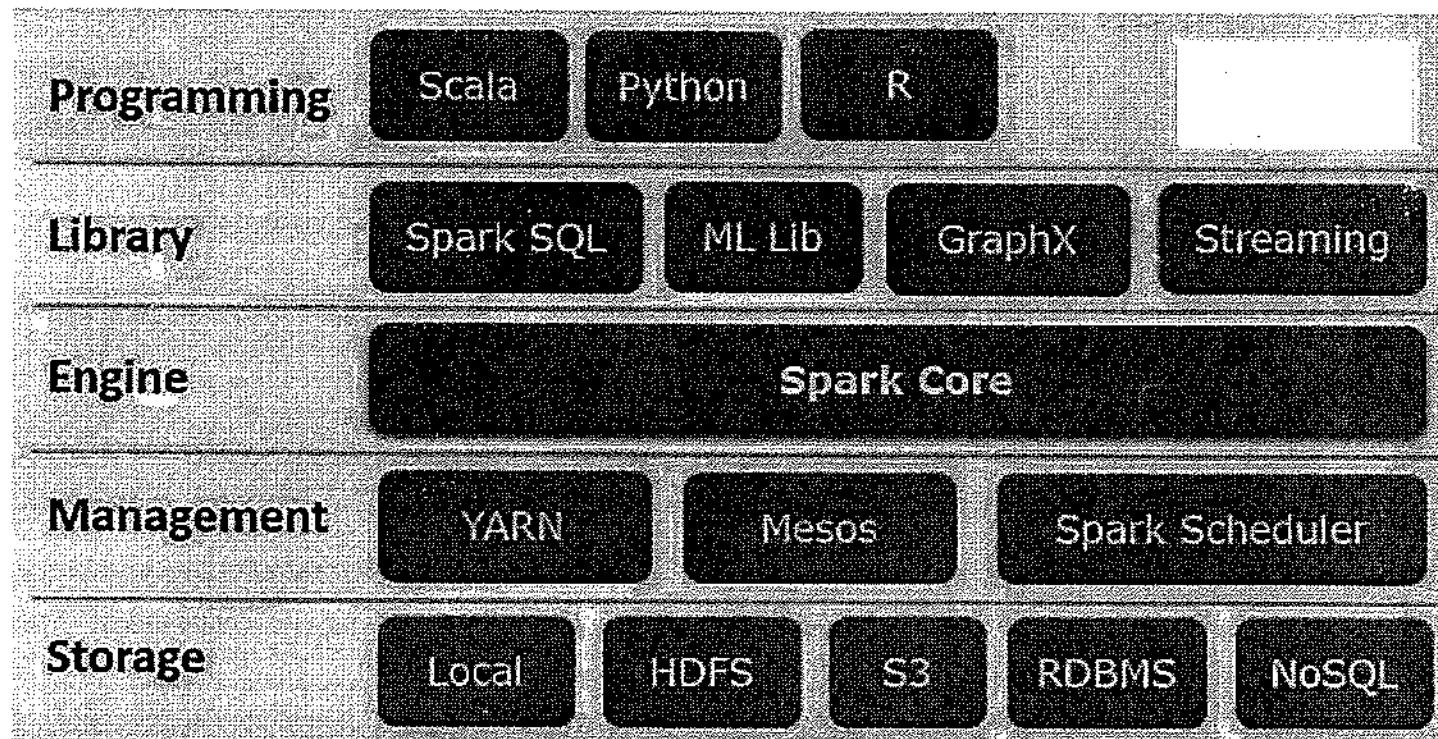
## Features of Spark

- *Speed* – Spark helps to run applications on Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- *Supports multiple languages* – Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages.
- *Advanced Analytics* – Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.





## Spark Architecture

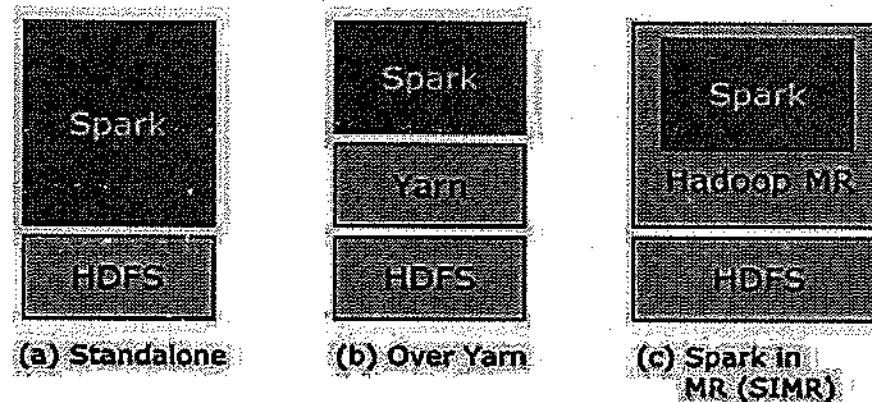


## Spark Components

- **Spark Core** – Underlying general execution engine to provide in-memory computing and referencing datasets in external storage systems. Responsible for basic I/O functionalities, Scheduling and Monitoring of jobs in Spark Clusters.
- **Spark SQL** – A component on top of Spark Core for working with structured data. It allows querying capabilities via SQL on Hive tables and other external data sources.
- **Spark Streaming** – Allows developers to perform batch processing and streaming of data in same application. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.
- **MLlib (Machine Learning Library)** – Provides multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering, as well as supporting functionality such as model evaluation and data import.
- **GraphX** – Data scientists can work with graph and non-graph sources to achieve flexibility and resilience in graph construction and transformation
- **SparkR** – R programming language has rich environment for machine learning and statistical analysis which helps increase developer productivity. Data scientists can now use R language along with Spark through SparkR for processing data that cannot be handled by a single machine.



## Spark Deployment Modes



**Standalone** – The Hadoop cluster can be equipped with all the resources statically and Spark can run with MapReduce in parallel. This is the simplest deployment.

**Over YARN** – Spark can be executed on top of YARN without any pre-installation. This deployment utilizes the maximum strength of Spark and other components.

**Spark in MapReduce (SIMR) / Mesos** – If you don't have YARN, you can also use Spark along with MapReduce. This reduces the burden of deployments.



## Resilient Distributed Datasets (RDD)

RDD is a fundamental data structure of Spark. Formally, RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

There are two ways to create RDDs – parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations.

## Advantages of Spark over native MapReduce Programming

If you want to use output of one MR application as input of another MR application, you have to store the output of first MR program to the HDFS, then only the other MR application can read data from HDFS and process. Data sharing is slow in MR due to replication, serialization, and disk IO. Both Iterative and Interactive applications require faster data sharing across parallel jobs.

When it comes to Spark, Spark stores the output of one application (in the form of RDD) into Distributed Memory instead of HDFS/Disk to make the process faster. If the Distributed memory (RAM) is not sufficient to store intermediate results (State of the JOB), then it will store those results on the disk. So, memory requirements for Spark is high when compared to native MR processing.

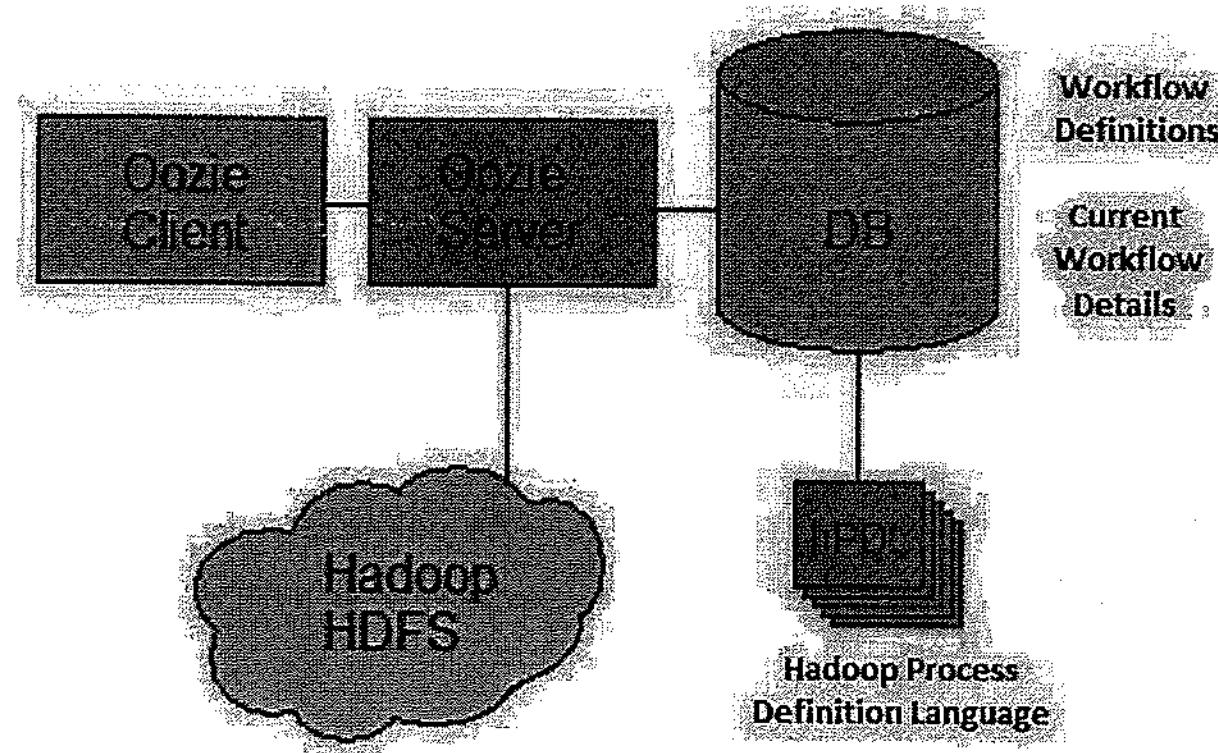


## Oozie

- Open-source Apache Project
- Initially developed by **YAHOO!**
- Oozie workflow jobs are **Directed Acyclical Graphs (DAGs)** of actions
- Oozie can run jobs sequentially (one after the other) and in parallel multiple jobs at a time
- Oozie can rerun the jobs from last point of failure
- Flexible to Start, Stop, Suspend and Rerun jobs
- Job execution will start only after all dependencies are met
- Allows you to restart the job from the point it is failed
- Tightly integrated with Hadoop security
- Oozie UI makes it easier to drill down to specific errors in the data Nodes



## Oozie Architecture





## Types of Oozie Jobs

- **WORKFLOW** jobs are DAGs, specifying a sequence of actions to execute
- **COORDINATOR** jobs are recurrent workflow jobs triggered by time (*frequency*) and data availability
- **BUNDLE** Jobs provide a way to package multiple coordinator and workflow jobs and manage life cycle of those jobs

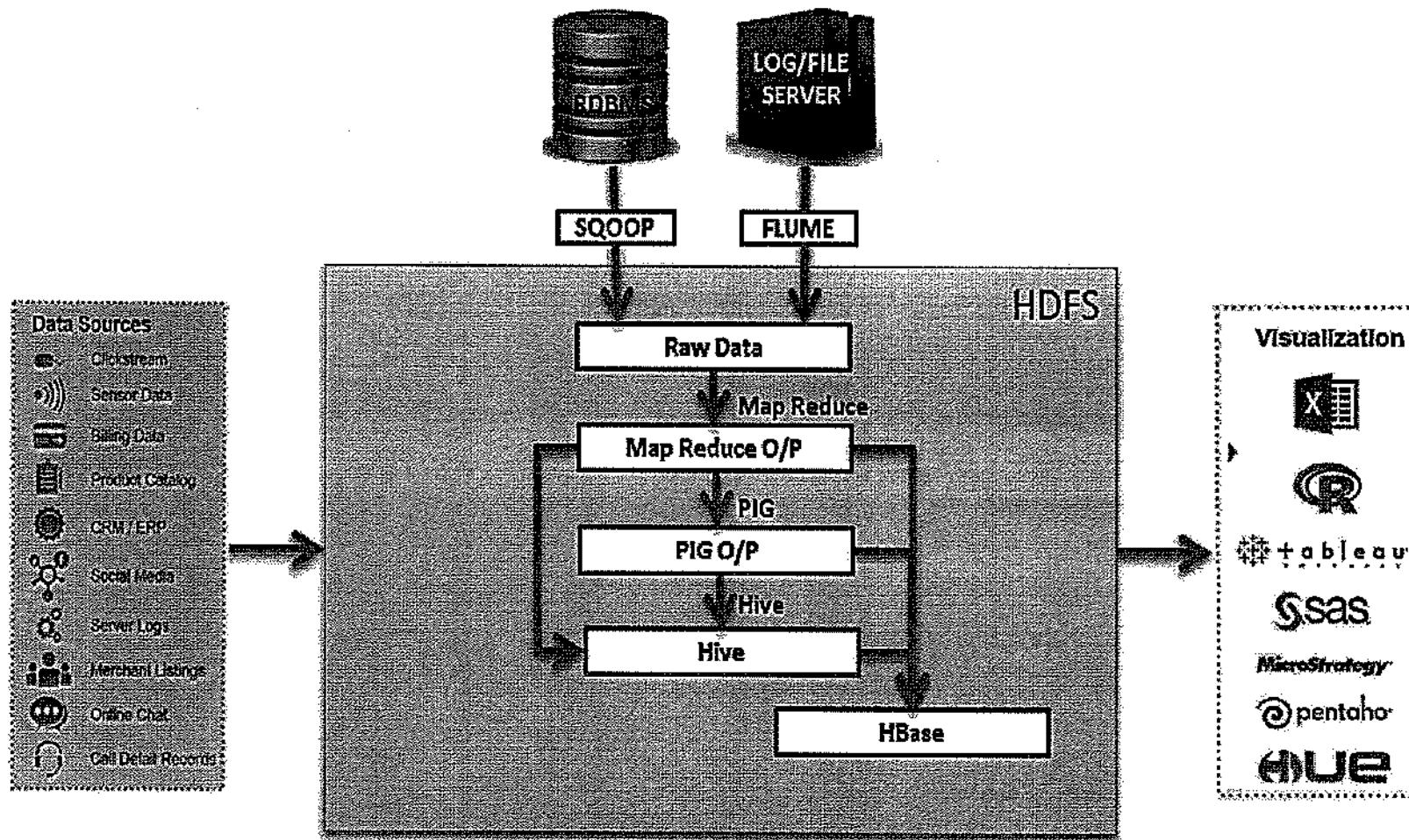


## Realtime issues from Oozie

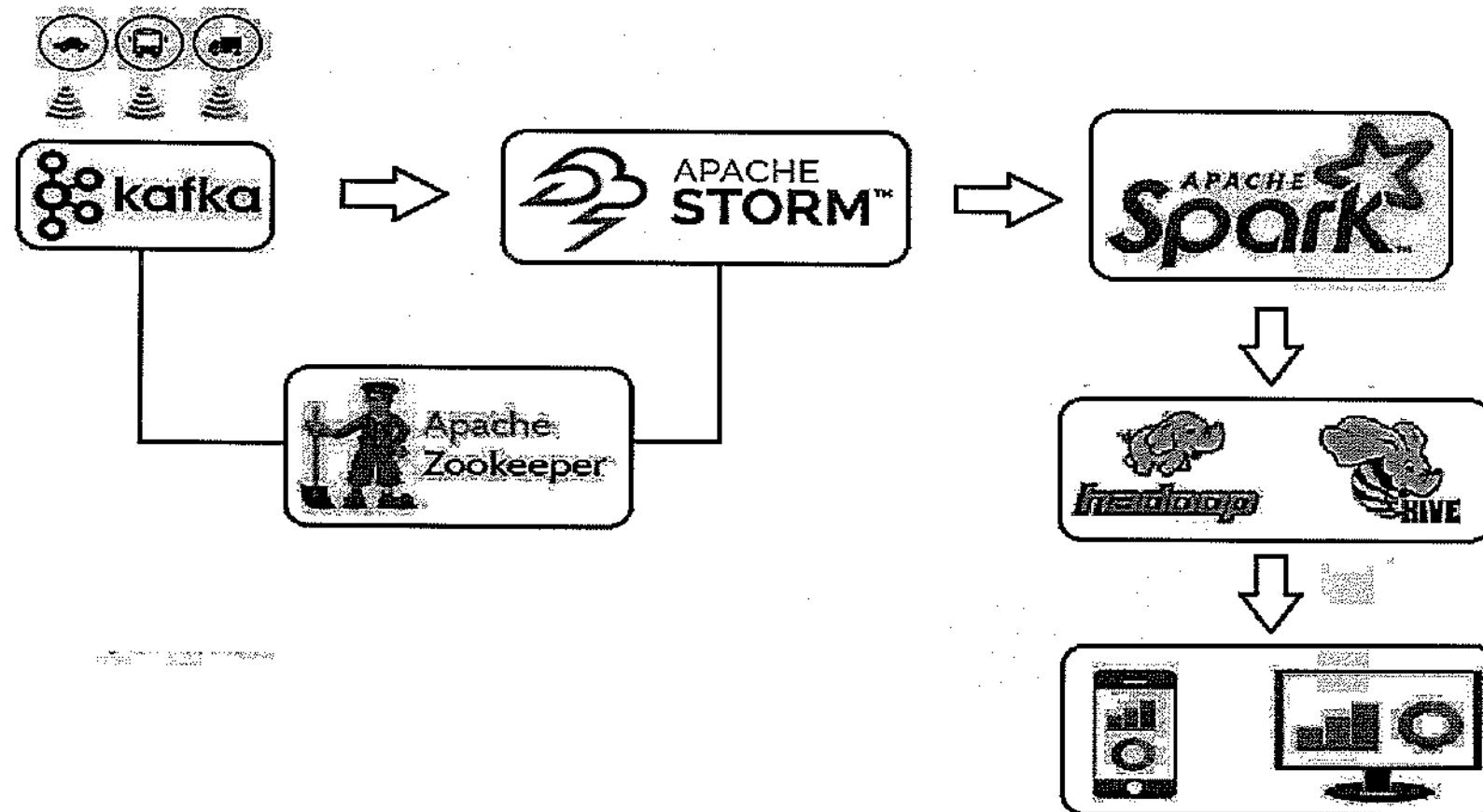
- If no Oozie HA, and Job History Server is down, Oozie is unable to submit the jobs
- Oozie Server will work like either MRV1 or YARN. Not both in the same environment
- When you use Hive Server 2 from Oozie, Oozie won't collect the Hadoop Job IDs of any jobs launched by Hive Server 2. So, troubleshooting will be difficult
- Token expiry issues
- Hadoop tools' libraries mismatch



## Sample Real-time Project Architecture with Batch Processing

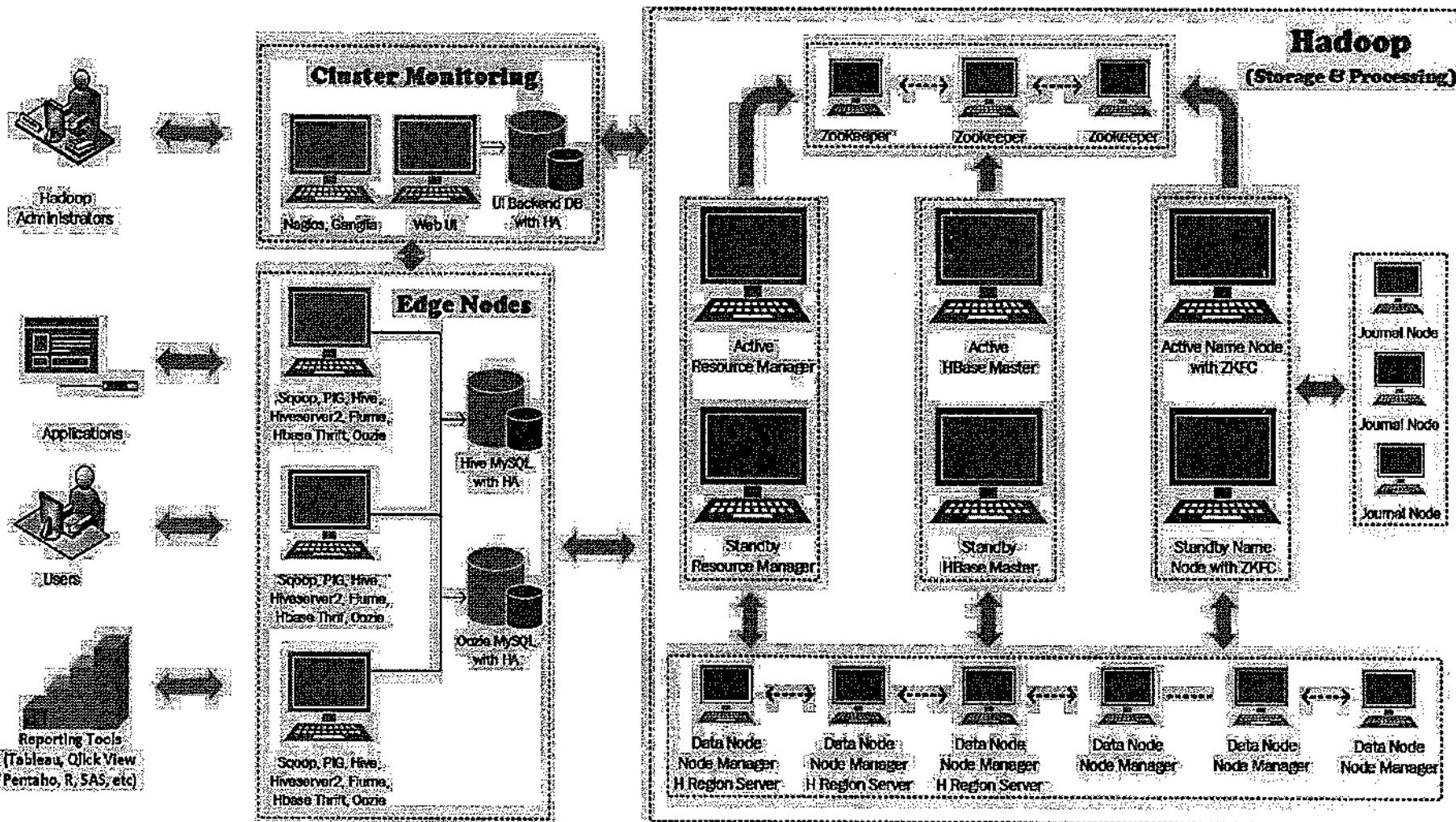


## Sample Real-time Project Architecture with Realtime Processing



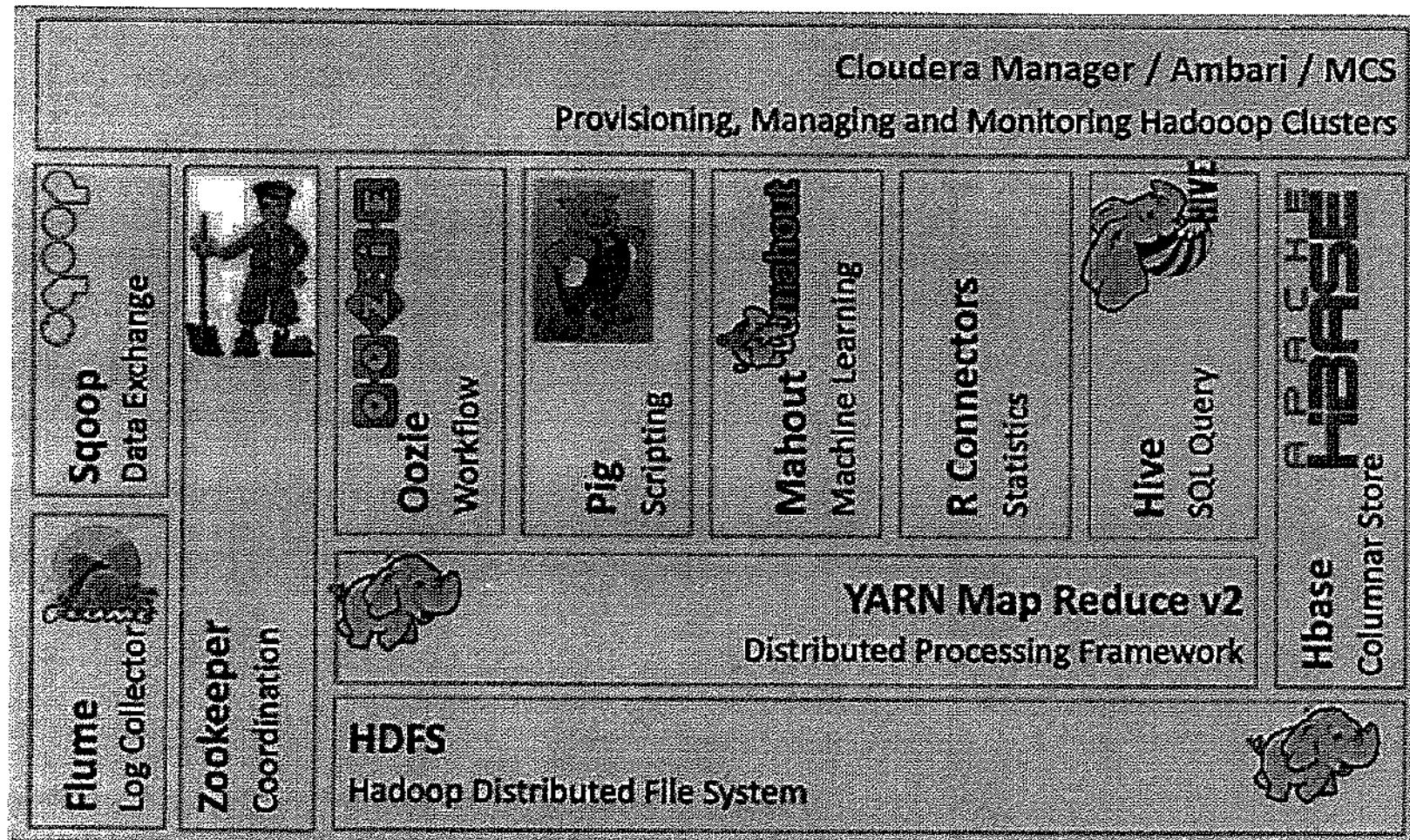


## Sample Real-time Cluster





## Hadoop Ecosystem Components





## Components that has High Availability in Hadoop Eco System

- ZooKeeper
- Journal Node
- NameNode
- Resource Manager
- MR Job History Server
- HiveServer2
- Hive Metastore Service
- HBase Master
- Flume Agents
- Oozie Server
- Spark History Server
- MySQL Databases which we use for CM/Ambari, Hive, Oozie, etc.



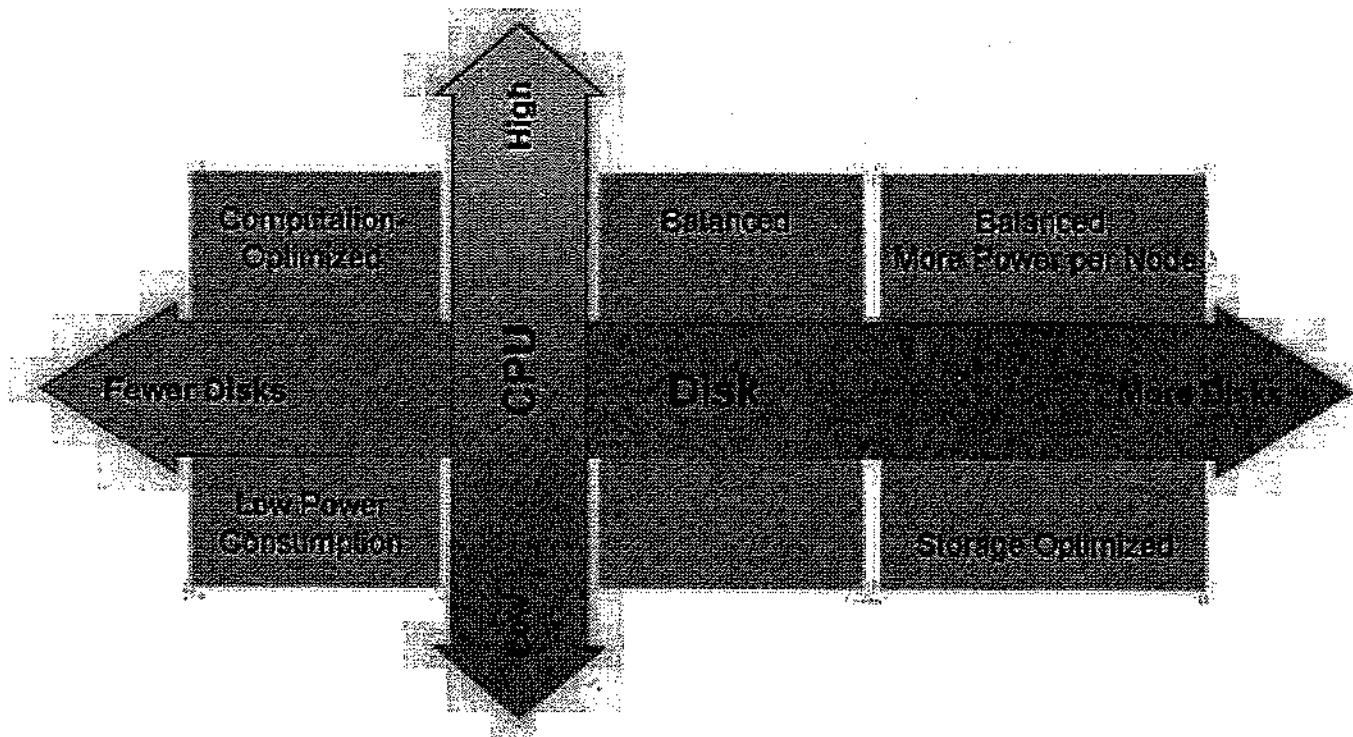
## Hadoop Ecosystem Components with latest stable versions

Ecosystem	Latest Stable Version		
Components/Tools	Cloudera	Hortonworks	MapR
CM/Ambari/MCS	5.13.3	2.6.1.3	5.2.0
CDH/HDP/MapR	5.13.3	2.6.4.0	5.2.0
Operating System	RHEL & Centos 7	RHEL & Centos 7	RHEL & Centos 7
MySQL	5.7, 5.6, 5.5, 5.1	5.7, 5.6, 5.5, 5.1	5.7, 5.6, 5.5, 5.1
JDK	JDK 1.8	JDK 1.8	JDK 1.8
Apache Hadoop	2.6.0	2.7.3	2.7.0
Apache Sqoop	1.4.6	1.4.6	1.4.6
Apache Pig	0.12.0	0.16.0	0.15.0
Apache Hive	1.1.0	1.2.1 & 2.1.0	1.2.0
Apache HBase	1.2.0	1.1.2	1.1.0
Apache Oozie	4.1.0	4.2.0	4.2.0
Apache Spark	1.6.0	1.6.3 & 2.2.0	1.6.0
Apache Kafka	0.9.0	0.10.1	
Apache ZooKeeper	3.4.5	3.4.6	3.4.5
Apache Flume	1.6.0	1.5.2	1.5.2





## Cluster Planning & Scaling





## CPU

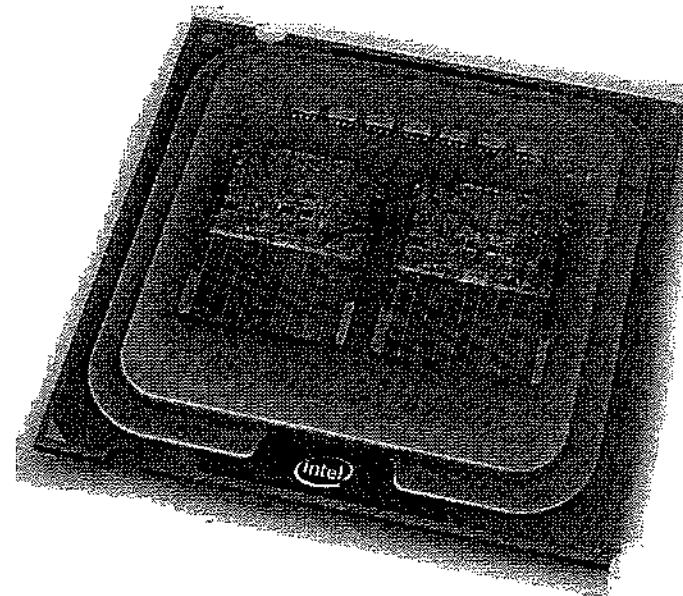
**More COREs means more resources**

**Preferably,**

**Development Cluster – 16 Cores**

**UAT / Test Cluster – 24 Cores**

**Production Cluster – 32 Cores or above**



## **RAM**

### ***NameNode***

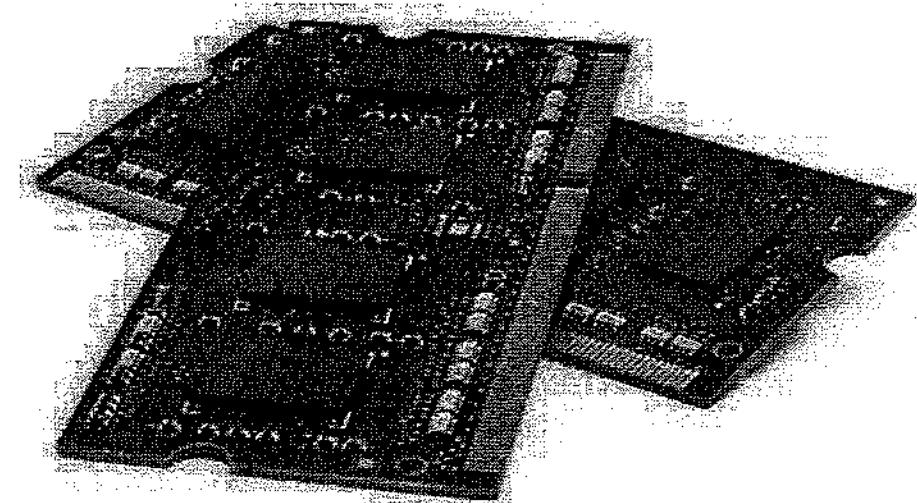
- RAM to fit metadata
- Roughly, 1GB for 1 Million Blocks

### ***Resource Manager***

- Retain Job Data, Memory Hungry
- Memory Requirements can grow independent of Cluster size

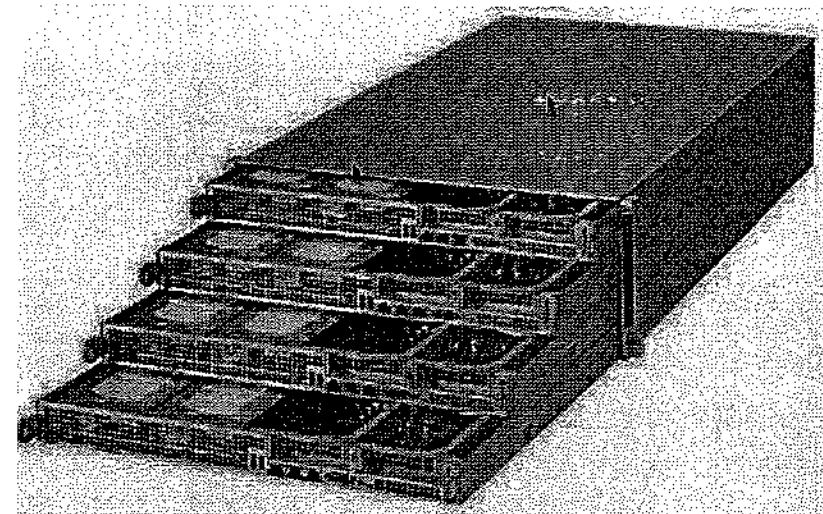
### ***DataNode***

- Performs both DataNode and NodeManager roles
- The more RAM, the more resources (containers)



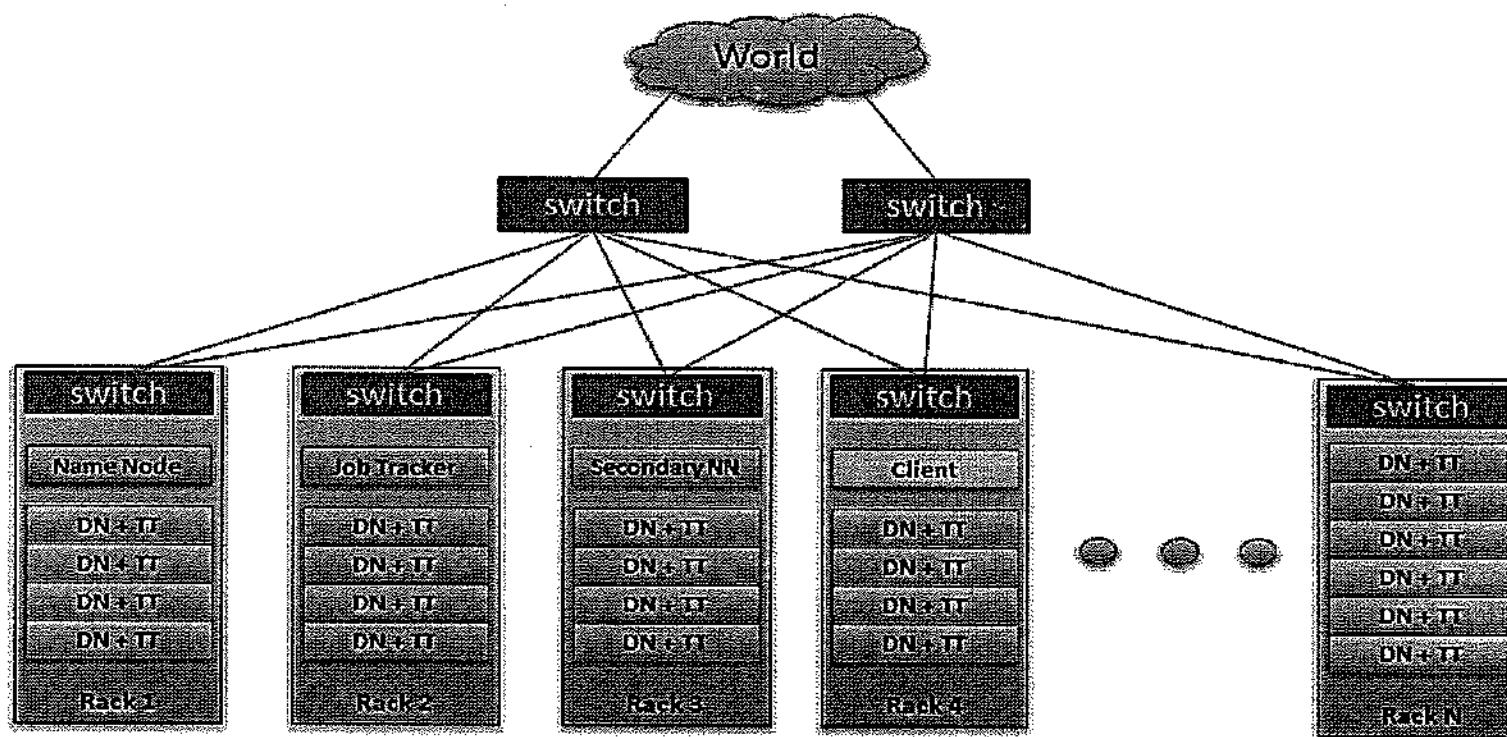
## **Hard Disk**

- Better to have RAID (Redundant Array of Independent Disks) for both OS & metadata disks (*to minimize the OS failures*)
- No RAID required to store HDFS data as we have *Replication* in place
- Better to have more disks for data distribution
- Better to have less data on 1 disk/host
- Need to have *good RPM* i.e. **10000 and above**



## Network

- Better to have *dedicated VLAN / Network*
- Better to have more than *one LAN / Network Interface Card* (1 card for each switch for HA)
- Better to have *10 G/Sec Ethernet cable*





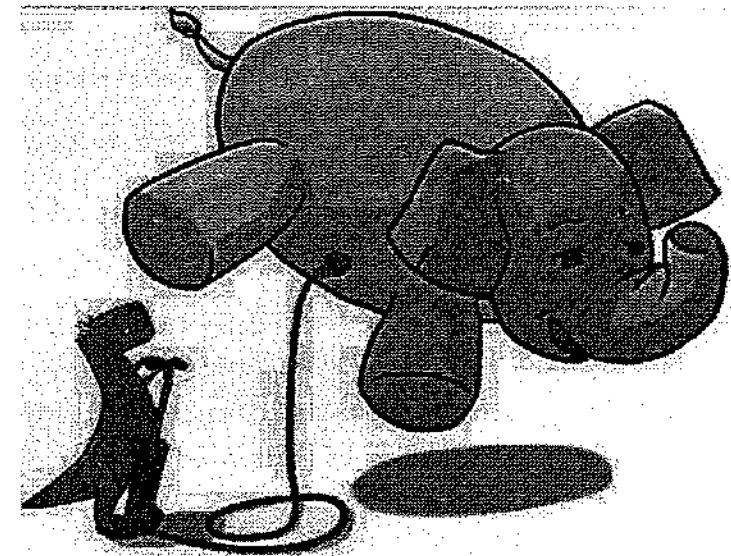
## Commonly used Node Configuration in Hadoop

	Development Cluster	UAT / Test Cluster	Production Cluster
CPU Cores	8	16	32
	16	32	40
	32	40	56
RAM	32 GB	64 GB	128 GB
	64 GB	128 GB	256 GB
	128 GB	256 GB	512 GB
Hard Disk	4 TB	8 TB	16 TB
	8 TB	16 TB	25 TB
Network	1 G/Sec	1 G/Sec	10 G/Sec
		10 G/Sec	



## **Plan your Cluster growth**

- It is important to understand the data growth
- If data grows at 10 TB/week, how many Nodes need to buy?
- What is Data Retention Period?
- How many cores required to process data?





## Hadoop Cluster Upgrade

Cluster Upgrade is required when you want to move from an older version of Hadoop to newer version of Hadoop.

An upgrade of HDFS makes a copy of the previous version's metadata and data. Doing an upgrade *does not double the storage requirements of the Cluster*, as the DataNodes use *hard links to keep two references* (for the current and previous version) to the same block of data. This design makes it straightforward to roll back to the previous version of the file system, should you need to. You should understand that *any changes made to the data on the upgraded system will be lost after the rollback completes*.

You can keep only the previous version of the file system: *you can't roll back several versions*. Therefore, to carry out another upgrade to HDFS data and metadata, you will need to delete the previous version, a process called finalizing the upgrade. Once an upgrade is finalized, there is no procedure for rolling back to a previous version.

Software numbering example: *Hadoop 2.6.4*

<b>2 – Major Version Number</b>	(changes every 12 to 18 months)
<b>6 – Minor Version Update</b>	(changes every three to four months)
<b>4 – Maintenance Release</b>	(changes when necessary)





## Types of Upgrade

- *Express Upgrade*

In Express Upgrade, the entire Cluster will be brought down and upgrade all Cluster nodes at a time. This incurs Cluster downtime and takes less time to complete full Cluster upgrade.

- *Rolling Upgrade*

Rolling upgrade feature takes advantage of the NameNode & Resource Manager high availability to enable you to upgrade your cluster software and restart the upgraded services without taking the entire cluster down. You must have NameNode & Resource Manager high availability enabled to perform a rolling upgrade. Since the upgrade happens over single node or on a set of nodes in serial process, this will take more time complete full Cluster upgrade.





## Cloudera Hadoop Cluster Upgrade

- Run the Host Inspector (fix any issues)
- Run the Security Inspector (fix any issues)
- Run `hdfs fsck /` and `hdfs dfsadmin -report` (fix any issues)
- Reserve a maintenance window before starting the upgrade to avoid alerts during the upgrade
- Take backup of Cloudera Manager/Hive/Oozie backend database(s)
- Take backup of NameNode Metadata & Zookeeper data directories
- Point the parcel location from Parcel Settings
- Perform upgrade of Cloudera Manager Server & Agents
- Perform upgrade of CDH with the help of '*Upgrade Cluster*' wizard
- Start all the services and make sure the Cluster is green
- Perform required sanity checks to fix any issues after upgrade
- Run '*Finalize Metadata Upgrade*' to finalize Cluster upgrade
- Exit maintenance mode once upgrade completed





## Hortonworks Hadoop Cluster Upgrade

- Run Service Check under Service Actions menu (fix any issues)
- Clear all alerts and make sure Cluster is green (fix any issues)
- Run `hdfs fsck /` and `hdfs dfsadmin -report` (fix any issues)
- Reserve a maintenance window before starting the upgrade to avoid alerts during the upgrade
- Take backup of Ambari/Hive/Oozie backend database(s)
- Take backup of NameNode Metadata & Zookeeper data directories
- Perform upgrade of Ambari Server & Agents
- Point the new version/stack details in '*Manage Versions*' wizard
- Perform upgrade of HDP with the help of '*Upgrade Options*' wizard
- Select either Rolling Upgrade or Express Upgrade
- Start all the services and make sure the Cluster is green
- Perform required sanity checks to fix any issues after upgrade
- Exit maintenance mode once upgrade completed



## Hadoop Distcp

**Distcp** is distributed copy. It launches MapReduce to copy huge data across same Cluster and different Clusters. Files previously copied will be skipped. The only check for duplicate files is that the file's name, size, and checksum are identical.

### ***Same Cluster:***

**hadoop distcp file1 file2**

**hadoop distcp dir1 dir2** (if dir2 exists, dir1 will be copied under it)

**hadoop distcp -update dir1 dir2**      (Updates only changes)

**hadoop distcp -overwrite dir1 dir2**      (Overwrite all files)

```
hadoop distcp hdfs://nn1.Cluster.com:9000/projects hdfs:// nn1.Cluster.com:9000/out
```

### **Across Clusters:**

```
hadoop distcp hdfs://nn1.hadoop.com:9000/foo/bar hdfs://nn2.hadoop.com:9000/foo/bar
```



## Hadoop Cluster Backup

### Failure Scenarios

- **Hardware Failures**
  - Data Corruption on Disk
  - Disk / Node Crash
  - Rack Failure
- **Backend database (MySQL, Oracle, etc.,) Failures**
- **User / Application Errors**
  - Accidental or Malicious Data Deletion
- **Data Center Failures**
  - Temporary Site Loss (due to Network, Power, etc.)
  - Permanent Site Loss (due to Fire, Ice, etc.)





## Data Corruption on Disk

- Checksum value for each block will be stored in `.meta` file
- If checksum does not match, NameNode discards block and replaces with fresh copy

## Disk / Node Crash

- Hardware failure detected by heartbeat loss
- First two replicas always on different hosts
- NameNode automatically re-replicate the blocks without enough replicas

## Rack Failure

- Configure at least 3 replicas and provide rack information (`topology.script.file.name`)
- 1<sup>st</sup> & 2<sup>nd</sup> replicas will be always in separate racks
- 2<sup>nd</sup> & 3<sup>rd</sup> replicas will be always in a different Nodes in the same rack
- Never lose all data if entire rack fails because of Rack Placement Algorithm





## Metadata Failure in CM / Ambari / Hive / Oozie

- MySQL backups
- MySQL High Availability (Master-Master or Master-Slave)

## User / Application Errors

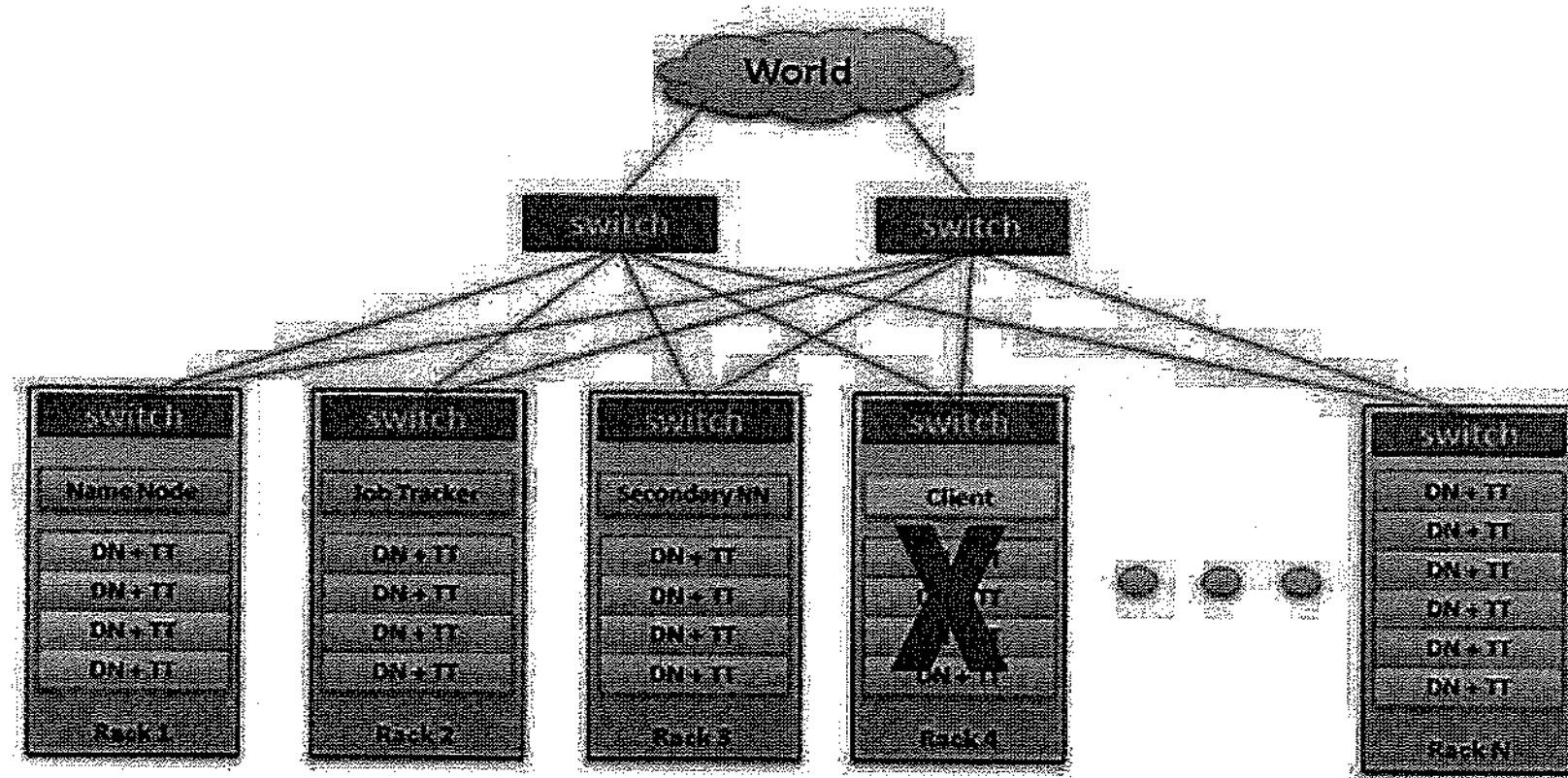
- Enable Hadoop Trash (make `fs.trash.interval > 0`)
- Enable `fs.trash.checkpoint.interval` to set trash cleaning

**NOTE: Moving to Trash will work only through FS shell. Programmatic deletes will not move data to Trash. Setup .Trash directory per user level.**



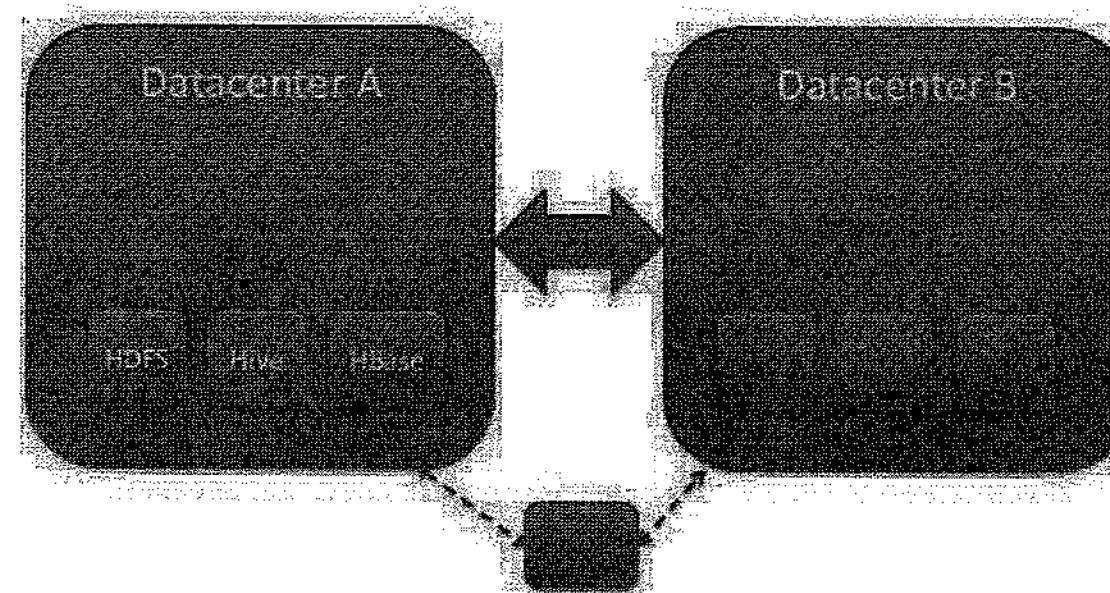


## Rack Failure





## Disaster Recovery



## HBase Data Backup & Recovery

- Full Shutdown Backup
  - Shutdown entire HBase Cluster (one of the disadvantage)
  - Use distcp command to copy entire /apps/hbase directory into backup Cluster
  - There are no chances of missing any in-flight changes to either StoreFiles or Metadata
  - HBase data can restored by using same distcp command when required
- Backup with *Export* Utility (In the same Cluster)
  - Built-in HBase utility to enable exporting of data into sequence files in HDFS
  - Runs MR job which makes a series of HBase API calls to get each row of data from HBase table
  - Enables incremental backups
- Live Cluster Backup using *Snapshots* (In the same Cluster)
  - Snapshots creates a moment-in-time for your tables by creating equivalent of UNIX hard links to your table's storage files on HDFS
  - Data is not duplicated at all but merely cataloged in small metadata files which will allow the system to roll-back to that moment-in-time state



## HBase Data Backup & Recovery (Cont'd)

- **Backup with *CopyTable* Utility (In the same or different Cluster)**
  - Built-in HBase utility which is very similar to *Export* utility
  - Key difference is that *CopyTable* writes its output directly to a destination table in HBase which is either in same Cluster or different Cluster
  
- **Live Cluster Backup using *Replication* (In different Cluster)**
  - Need to have two different Clusters to achieve Replication
  - Uses Source Cluster's WAL to maintain synchronization in the Destination Cluster
  - Replication has three modes: master->slave, master<->master, and cyclic
  - In the event of data center outage, Client applications can be redirected to other Cluster

## HDFS Snapshots

HDFS Snapshots are read-only point-in-time copies of the file system. Snapshots can be taken on a sub-tree of the file system or the entire file system. Snapshots handles data corruption from user/application and accidental deletes.

- Snapshots can be taken on any directory once the directory has been set as *snapshottable*
- A snapshottable directory is able to accommodate 65,536 simultaneous snapshots
- There is no limit on the number of snapshottable directories
- If there are snapshots in a snapshottable directory, the directory can be neither deleted nor renamed before all the snapshots are deleted
- Nested snapshottable directories are currently not allowed. In other words, a directory cannot be set to snapshottable if one of its ancestors/descendants is a snapshottable directory.
- For a snapshottable directory, the path component ".snapshot" is used for accessing its snapshots
- When upgrading from an older version of HDFS, existing paths named .snapshot need to first be renamed or deleted to avoid conflicting with the reserved path



## HDFS Snapshot Commands

**hdfs dfsadmin -allowSnapshot <path>**

**hdfs dfsadmin -disallowSnapshot <path>**

**hdfs dfs -createSnapshot <path> [<SnapshotName>]**

**hdfs dfs -deleteSnapshot <path> <SnapshotName>**

**hdfs dfs -renameSnapshot <path> <oldName> <newName>**

**hdfs lsSnapshottableDir**

**hdfs snapshotDiff <path> <fromSnapshot> <toSnapshot>**



## OS Patching

Like all OSes, every once in a while you need to update the software running on your Linux server. You can do this in one of three ways:

- Download the updated packages and manually install them yourself.
- Use a built-in open source application that comes with the OS distribution.
- Use a third-party application that downloads the file and then runs the installation for you.



## Manual Patching

### *RPM Installation / Update*

- Download the packages directly from the online file repository or a trusted mirror site
- Run “*rpm -ivh*” to install the new package, or “*rpm -Uvh*” if you are updating the existing package

## Patching using built-in tools

### *Local Repository (YUM - YellowDog Updater, Modifier)*

- Download the packages directly from the online file repository or a trusted mirror site and create a Local Repository
- Run “*yum install <package name>*” to install new package, or “*yum update <package name>*” to update the existing package



## Configuration Management Tools

- **Chef** is an open source tool for configuration management, focused on the developer side for its user base. Chef operates as a master-Client model, with a separate workstation needed to control the master. It's based in Ruby, with pure Ruby used for most elements you write.
- **Puppet** is based in Ruby, but uses a customized Domain Scripting Language (DSL) closer to JSON for working within it. It runs as a master-Client setup and uses a model-driven approach.
- **Ansible** is an open source tool used to deploy applications to remote Nodes and provision servers in a repeatable way
- **SaltStack** is a CLI-based tool that can be set up as a master-Client model or a non-centralized model. Based in Python, Salt offers a push method and an SSH method of communication with Clients. Salt allows for grouping of Clients and configuration templates to simplify the control of the environment.
- **StackIQ** is a Centos/RHEL bare metal install tool that can take your servers from bare hardware (or virtual hardware) to working Linux, ready to install applications. StackIQ does this at scale, so deploying 1000+ servers is no more complex than deploying one.





## Monitoring Hadoop Clusters

- Use of a monitoring tool is important to warn you of potential or actual problems on individual machines in the Cluster
- You can integrate Cluster monitoring into existing monitoring tools like *Nagios* and *Ganglia*
- *Cloudera Manager / Ambari / MapR Control System* provides comprehensive Cluster monitoring with no additional configuration required
  - Monitor Cluster health
  - Identify configuration issues
  - Track metrics and resource usage with charts and dashboards
  - View event logs
  - View audits
  - Generate alerts
  - Generate reports





## Nagios Mail Alert Sample

The screenshot shows an email client interface with various toolbar icons like Move, Rules, OneNote, Unread, Categorize, Follow, Read, Up, Search People, Address Book, Filter Email, Report Phishing, Find, and PhishMe. The email message is from [ham-nagios@hadoop.com](mailto:ham-nagios@hadoop.com) on Thu 2/23/2017 6:01 AM. The subject is "MyCluster (ham.hadoop.com) Hiveserver2 - CRITICAL (0d 0h 20s)". The recipient is "To: Hadoop Infrastructure Support Team". The notification type is "PROBLEM". The service is "Hiveserver2", host is "ham.hadoop.com", cluster is "MyCluster", and state is "CRITICAL". The date/time is "Wed Feb 22 17:30:40 MST 2017".

ham-nagios@hadoop.com  
MyCluster (ham.hadoop.com) Hiveserver2 - CRITICAL (0d 0h 20s)

To: Hadoop Infrastructure Support Team

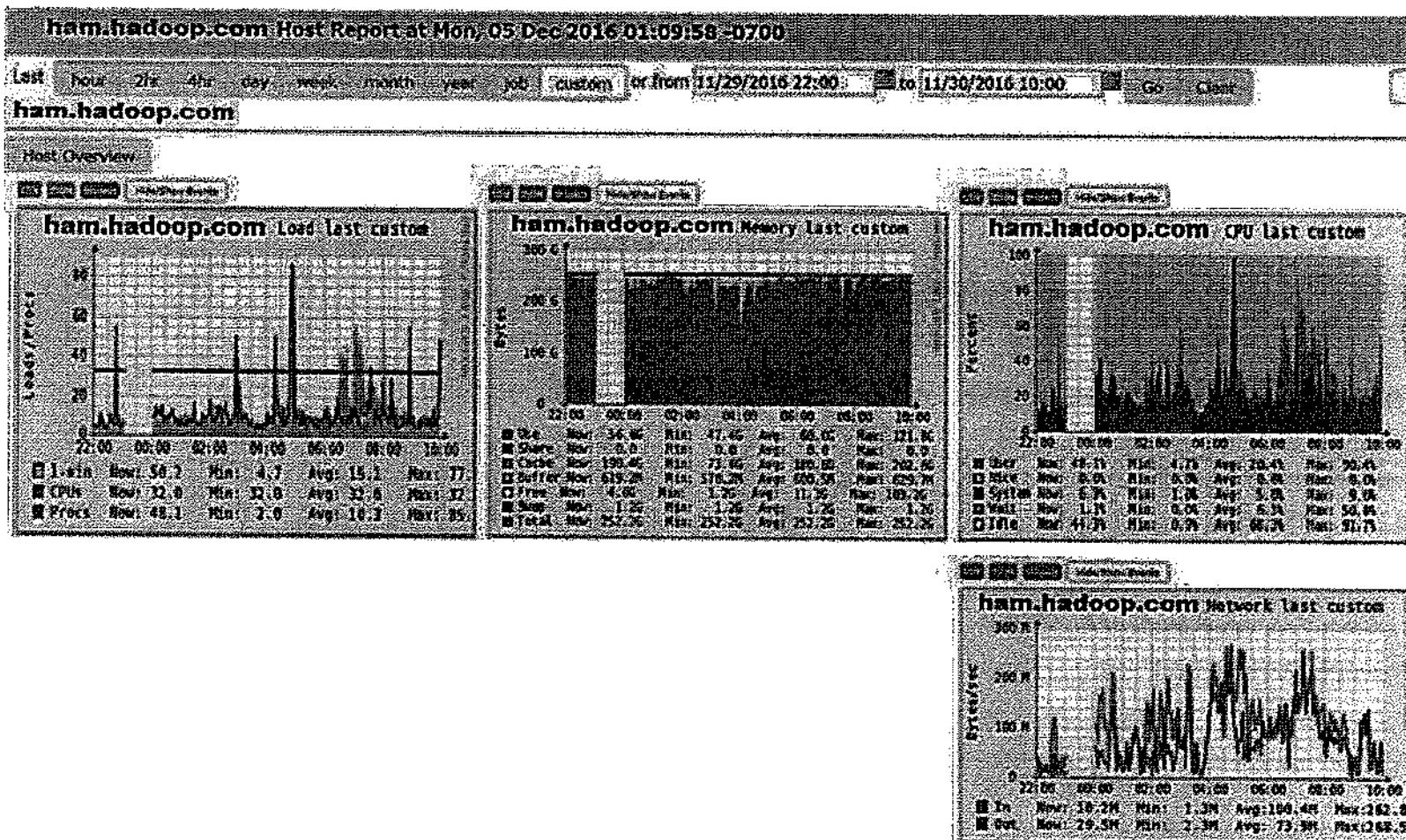
Notification Type: PROBLEM

Service: Hiveserver2  
Host: (ham.hadoop.com)  
Cluster: MyCluster  
State: CRITICAL

Date/Time: Wed Feb 22 17:30:40 MST 2017



## Ganglia Stats





# Security on Hadoop



## Hadoop Security

- ***Authentication***
  - Process of verifying the identity of a user or a service
  - User authentication via LDAP/AD
  - User/Service Authentication control via Kerberos
- ***Authorization (Access Control)***
  - Process of specifying access rights to users or services
  - HDFS has traditional POSIX-style permissions for files and directories
  - Access Control Lists (ACLs) for HDFS
- ***Auditing***
  - Audit Logs on hosts to record user's activities
- ***Data Encryption***
  - Local File System level
  - HDFS level
  - Network level

## Types of Hadoop Security

- ***HDFS File Ownership and Permissions***
  - Read, Write & Execute Permissions for User, Group & Others
  - Provides modest protection and less effective
  - Mainly intended to safeguard against accidental deletions
- ***HDFS Access Control Lists (ACL)***
  - ACL provides a way to set different permissions for specific users or groups, not only the file's owner and the file's group
- ***Enhanced security with Kerberos***
  - Provides strong authentication of both clients and servers
  - Wraps Hadoop API calls in an SASL handshake
  - Applications can be run under the submitter's own account
- ***Security of a Cluster is enhanced by isolation***
  - It should ideally be on its own network
  - Access to Nodes/network should be limited for un-trusted users
- ***Encrypted HDFS data transfers***





## HDFS Access Control Lists (ACL)

ACL provides a way to set different permissions for specific users or groups, not only the file's owner and the file's group. By default, support for ACLs is disabled. To enable ACLs, add below to *hdfs-site.xml*

```
<property>
  <name>dfs.namenode.acls.enabled</name>
  <value>true</value>
</property>
```

Examples:

```
hdfs dfs -getfacl /sample.txt
hdfs dfs -setfacl -m user:blruser:rw- /sample.txt
hdfs dfs -setfacl -m group:blrproj:rw- /sample.txt
hdfs dfs -setfacl -b /sample.txt
hdfs dfs -setfacl -R -m user:blruser:rw- /out
hdfs dfs -setfacl --set user::rw-,group::r--,other::r--,user:blruser:rw-,group:blrproj:rw- /sample.txt
```





## HDFS Access Control Lists (ACL) (Cont'd)

### **DEFAULT ACL:**

A default ACL may be applied only to a directory, not a file. All the child files and directories receive default ACL settings.

Ex:

```
hdfs dfs -mkdir /Rawdata  
hdfs dfs -setfacl -m default:group:blrproj:r-x /Rawdata  
hdfs dfs -getfacl /Rawdata  
hdfs dfs -mkdir /Rawdata/Jan  
hdfs dfs -getfacl /Rawdata/Jan
```

To block all access to */Rawdata* by user *adam*:

```
hdfs dfs -setfacl -m user:adam:--- /Rawdata
```



## What is Kerberos?

- Kerberos is a network authentication protocol created by Massachusetts Institute of Technology (MIT). It *eliminates the need for transmission of passwords across the network* and removes the potential threat of attackers over the network
- Kerberos is used to authenticate *Users* and *Services* both
- KDC – Key Distribution Centre
  - KDC is the authentication server in Kerberos environment. KDC can be logically divided into three parts.
    - *Database* - Stores users and services identities known as Principals. KDC database also stores information like an encryption key, ticket validity duration, expiration date etc.
    - *Authentication Server (AS)* - Receives authentication request from the Client and issues a special ticket known as Ticket Granting Ticket (*TGT*)
    - *Ticket Granting Server (TGS)* - TGS is the application server of KDC which provides service ticket



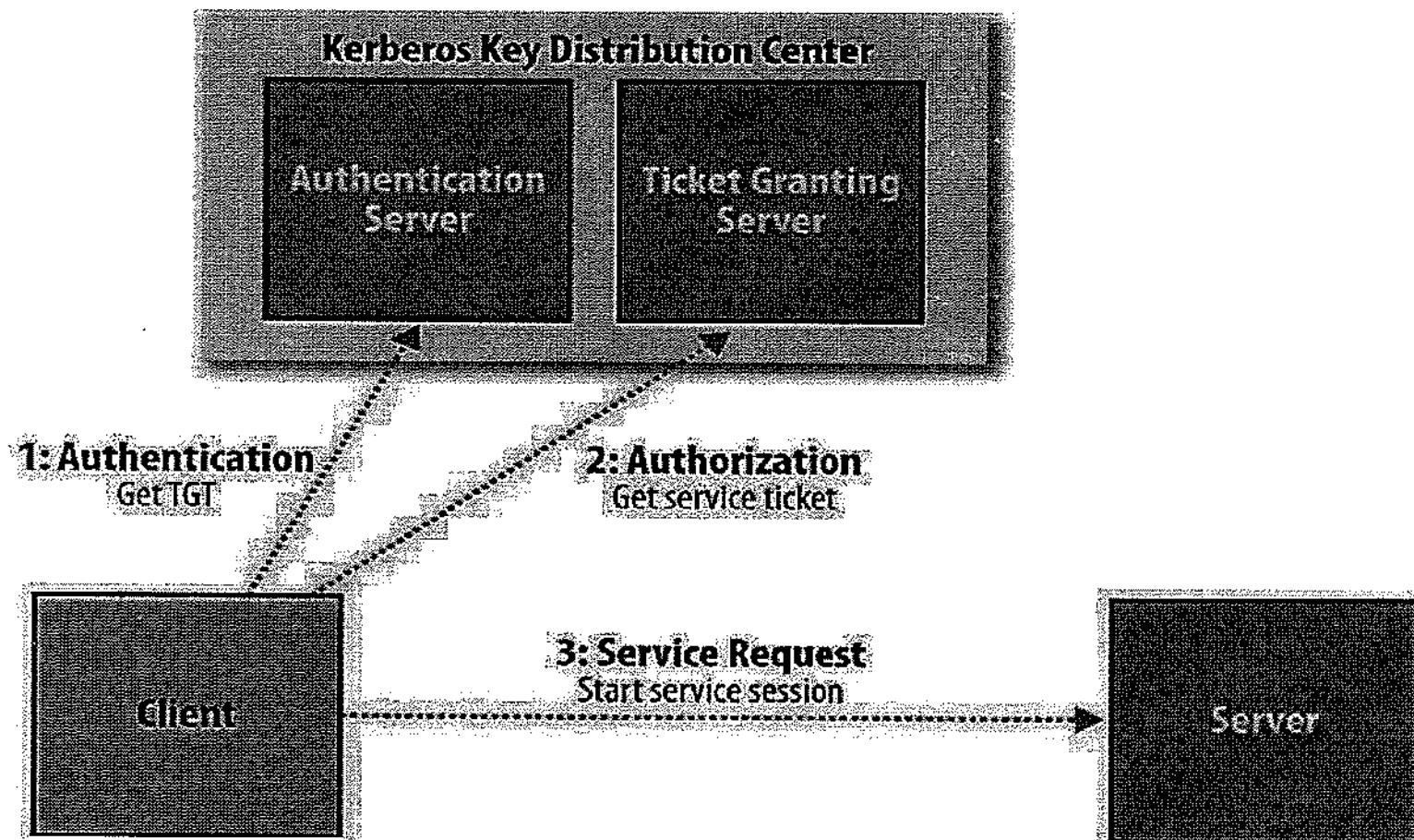
## **Kerberos Terminology**

**Knowing a few terms will help you in understanding Kerberos**

- **Realm**
  - A logical group of machines participating in a Kerberos network
  - Similar to a domain (**HADOOP.COM**) in DNS in *uppercase*
- **Principal**
  - A unique identity to identify User or Service
  - Made up of three distinct components: Primary, Instance, and Realm
    - Primary – Generally a User Name or Service Name
    - Instance – Defines the Host Name but optional
- Ex: **nn/hn1.hadoop.com@HADOOP.COM**
- **Keytab file**
  - A file that stores Kerberos principals and encrypted copy of their keys
  - Allows non-interactive access to services protected by Kerberos



## How Kerberos works?



## How Kerberos works? (Cont'd)

- User/Client initiates the authentication request (i.e. made with *kinit*)
- This request will be directed to Authentication Server (AS)
- Authentication Server (AS) reply to the user request and provides Ticket Granting Ticket (TGT)
- User/Client will go to Ticket Granting Server (TGS) with this TGT for a service request
- Ticket Granting Server (TGS) will provide the service request
- User/Client sends this service request to Application Server to access the service
- Timestamps are an essential part of Kerberos
  - Make sure you synchronize system clocks (*NTPD*)



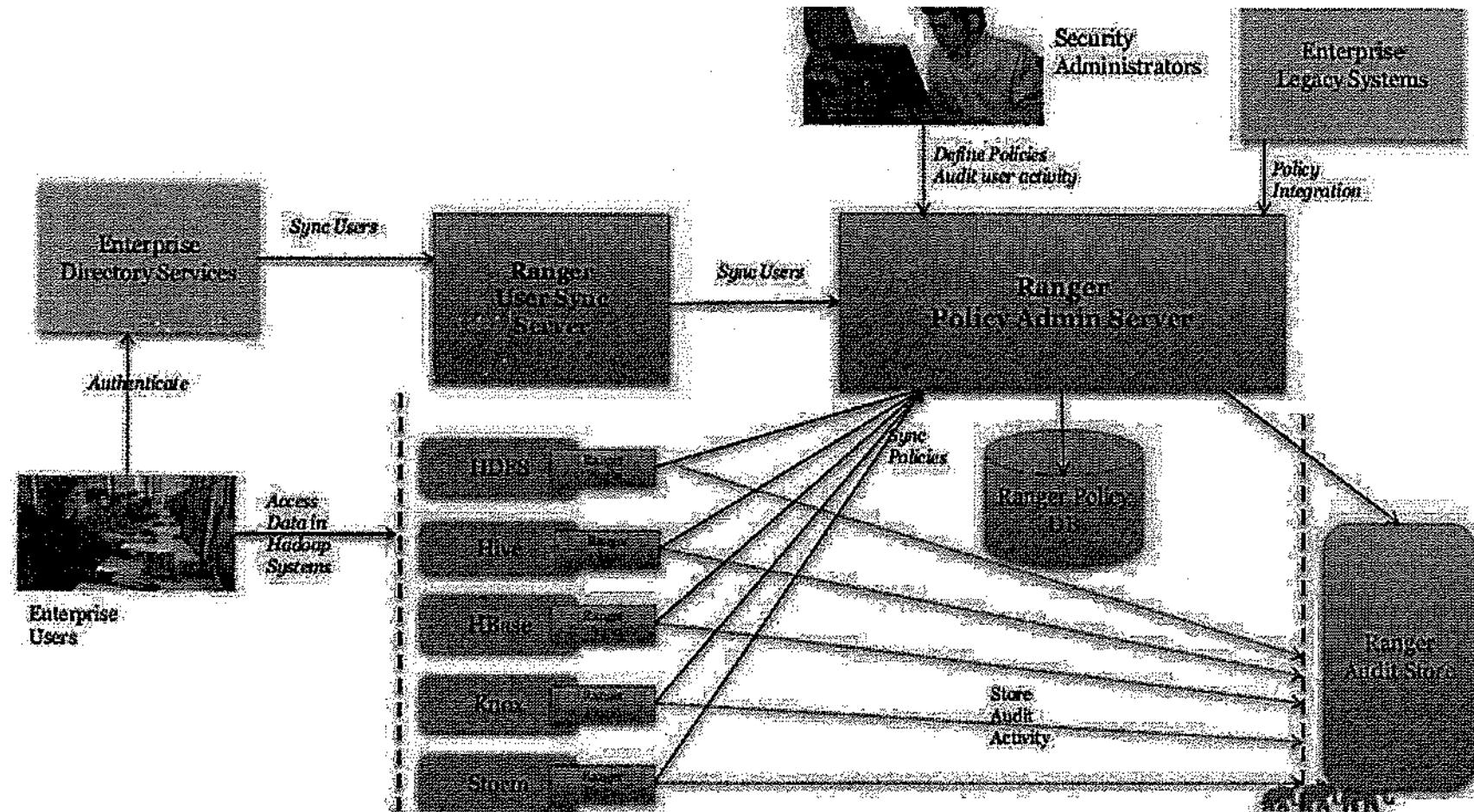


## Ranger

- Open-source *Apache Project*
- Centralized security framework to define, administer and manage security policies across Hadoop Components
- Security policies can be set to individual users or groups to access files, folders, databases, tables or columns
- Each Hadoop Component will be aligned with a Ranger Plugin
- Ranger currently supports Hadoop components like HDFS, YARN, Hive, HBase, Storm, Knox, Solr, Kafka, etc.
- Support for Transparent Data Encryption with KMS implementation



## Ranger Architecture





## Ranger Components

- **Ranger Policy Admin Server**
  - Central Interface for security administration to manage which users/groups can access to which Hadoop Components
  - User can create and update policies, view audit activities
- **Ranger User Sync Server**
  - Synchronization utility to pull users and groups from Unix or LDAP or Active Directory
  - User/Group information is stored within Ranger Admin Policy DB and used for policy definition
- **Ranger Plugin**
  - Lightweight Java program within Hadoop Component
  - Pulls all policies from Admin Server and store them locally in cache
  - Acts as authorization module and evaluate user requests against security policies before granting access
  - Collects data from user requests and store it into the Audit Store





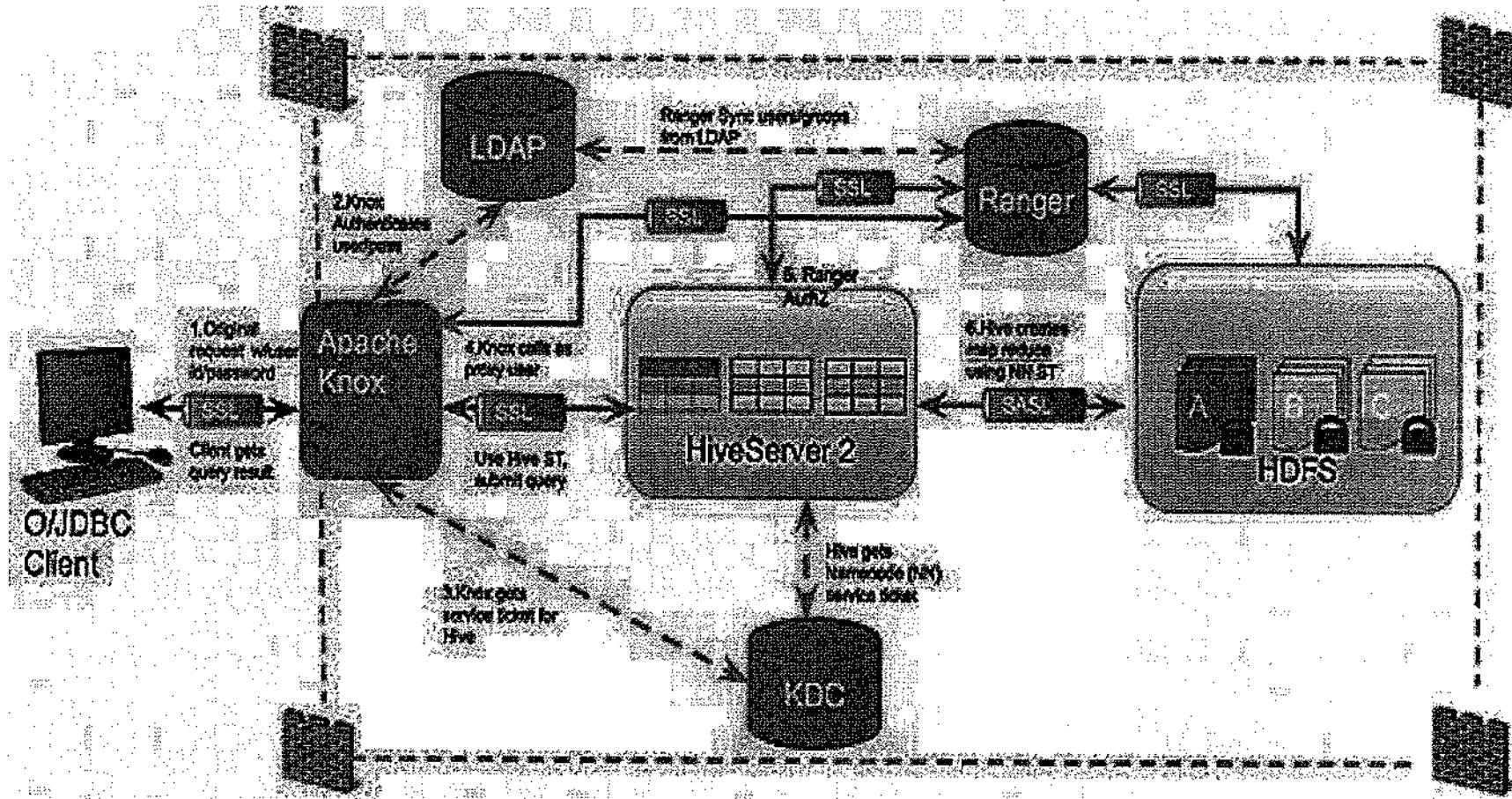
## Knox Gateway

- Open-source *Apache Project*
- Single and Secured *entry point* for Hadoop Clusters
- Centralized control system to route the outside requests to multiple Hadoop Clusters
- Can expose Hadoop services to users outside of Hadoop Cluster without reducing security
- Can be used with both *Kerberized* and *Non-Kerberized* Hadoop Clusters
- Enhances the Cluster security without revealing network and host details
- Enterprise Integration which supports *LDAP, AD, SSO and other authentication systems*





## Knox Gateway Architecture



## What is SSL, TLS & HTTPS?

**Secured Socket Layer (SSL)** is a standard security protocol for establishing encrypted links between a web server and a browser in an online communication. The usage of SSL ensures that all data transmitted between the web server and browser remains encrypted. SSL certificate is necessary to create SSL connections.

- Create a Java Key Store (JKS) by providing all details about the identity of your website, company details, and location. JKS contains 2 cryptographic keys – Public Key & Private Key
- Extract Certificate Signing Request (CSR – contains your details & Public Key) and send to Certification Authority (CA)
- CA team will provide the signed SSL certificates with validity
- Import the SSL Certs into JKS
- Point the JKS location and JKS password in the web server configuration file

**Transport Layer Security (TLS)** is just an updated, more secure, version of SSL.

**Hyper Text Transfer Protocol Secure(HTTPS)** appears in the URL when a website is secured by an SSL certificate. The details of the certificate, including the issuing authority and the name of the website owner, can be viewed by clicking on the lock symbol on the browser bar





## How does SSL work?

- Browser attempts to connect to a web server (ex: webhdfs, hiveserver2, Knox etc.,) which is secured with SSL
- To initiate secure connection, the web server identifies itself by sending a copy of its SSL Certificate
- Browser validate whether the SSL is signed by trusted CA, and it is not expired and it contains correct domain details
- Once everything is validated, browser creates a symmetric session key by encrypting with public key of the SSL certificate
- Web server uses its private key to decrypt the symmetric session key and sends an acknowledgement that is encrypted with the session key

**From now on, all data/requests transmitted between the web server and the browser is encrypted and secure**



## Common Sources of Problems

- Configuration related Issues
  - Start with recommended configuration values and
  - Don't rely on Hadoop defaults completely
  - Understand the precedence of overrides
  - Control your Clients' ability to make configuration changes
  - Test changes before putting them into production
  - Look for changes when deploying new releases of Hadoop
  - Automate configuration management
  - Benchmark systems to understand their impact on your Cluster
- Resource exhaustion
  - Not enough Cores
  - Not enough RAM
  - Not enough Disk
  - Not enough Network Bandwidth





# Hadoop Cluster Performance Tuning in Operating System level



## Operating System & Network Tuning

- Disable transparent Huge Page Compaction
  - Latest versions of Linux support memory block sizes (pages) of 2MB and 1GB along with 4KB blocks. If THP is enabled and system doesn't get continuous 2MB blocks, performance will degrade.
  - `sudo sh -c "echo never > /sys/kernel/mm/transparent_hugepage/defrag"`
- `vm.swappiness` - Linux moves memory pages that have not been accessed for some time to the swap space even if there is enough free memory available. Value should be less than or equal to 10.
- Disable power saving options in BIOS (`setterm -blank 0 -powersave off -powerdown 0`)
- `ext4` or `xfs` are reliable and recommended. Mount your local disks using `noatime` option to speed up read capabilities
- Set number of open files in parallel option to avoid "*Too many open files*" exceptions during job execution. By default, it is 1024. Change this to max of 32768 by using `ulimit` command
- Turn off caching on disk controller (Disk cache holds data that has been read recently)
- `net.core.somaxconn = 1024` (default socket listen queue size 128)
- Maximum Transmission Unit (MTU) indicates the size of packet that can be sent over TCP. By default, it is 1500 bytes. Set this to a big number as MTU=10240 in `ifcfg-eth` file.

## Operating System & Network Tuning (Cont'd)

- Linux Kernel Parameters

- **fs.file-max=6815744** (Total number of file descriptors)
- **fs.aio-max-nr=1048576** (Maximum number of concurrent I/O requests)
- **net.core.rmem\_default=262144** (Default OS receive buffer size)
- **net.core.wmem\_default=262144** (Default OS send buffer size)
- **net.core.rmem\_max=16777216** (Max OS receive buffer size)
- **net.core.wmem\_max=16777216** (Max OS send buffer size)
- **net.ipv4.tcp\_rmem=4096 262144 16777216**  
(Minimum, Default and Maximum receive window size)
- **net.ipv4.tcp\_wmem=4096 262144 16777216**  
(Minimum, Default and Maximum send window size)



# Hadoop Cluster Performance Tuning in Storage Layer



## Data Disk Scaling

- Better to have more number of small disks rather than less number of large disk to improve disk I/O performance
- Use multiple disk mount points on slave Nodes (use JBOD configuration not RAID)
- Better to use 10000/12500 RPM disks or more
- Can use SATA drives, thereby keeping the overall cost of the server as low as possible
- Make sure the minimum size of file is at least a block size on HDFS to save disk space
- Mount disk volumes with *noatime*



## Applying HDFS Quotas

We have 2 different quotas: Name Quota & Space Quota

`hadoop dfsadmin -setQuota <N> <directory>`

`hadoop dfsadmin -clrQuota <directory>`

`hadoop dfsadmin -setSpaceQuota 1m`

`hadoop dfsadmin -setSpaceQuota <N>`

`hadoop dfsadmin -clrSpaceQuota`





# Hadoop Cluster Performance Tuning in Processing Layer



## Submitting Jobs to Queues

- Users submit jobs to Queues
- Queues, as collection of jobs, use ACLs to control which users can submit jobs to them
- Hadoop comes with a single mandatory queue, called “*default*”
- Queues are expected to be primarily used by Hadoop Schedulers
- Queue names are defined in the `<mapreduce.job.queuename>` property
- If no Queue names are defined, job will be submitted to “*default*” queue





## Hadoop Job Schedulers

**FIFO Scheduler**      **(By default with Apache 1.X)**

**Fair Scheduler**      **(By default with Cloudera & MapR)**

**Capacity Scheduler**    **(By default with Apache 2.X & Hortonworks)**

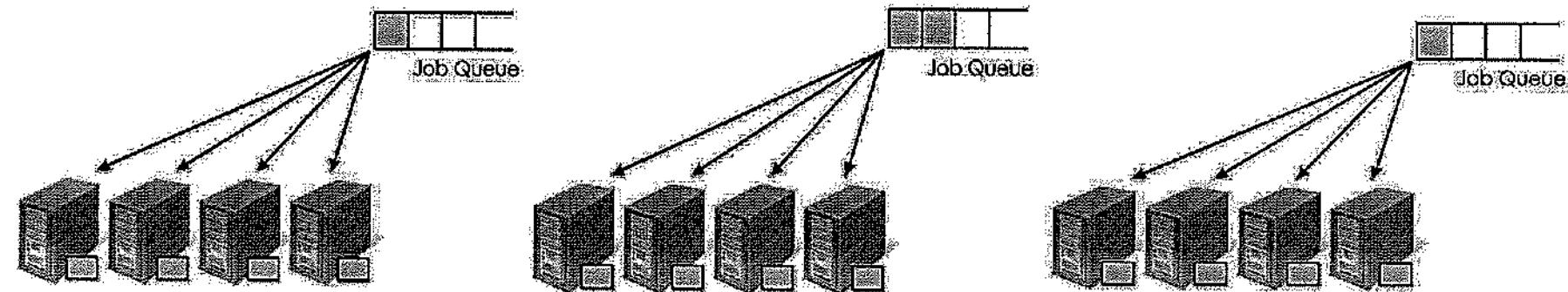


## FIFO Scheduler

The user who submits the job first, gets to use the Cluster first. Other Jobs wait in queue for their turn.

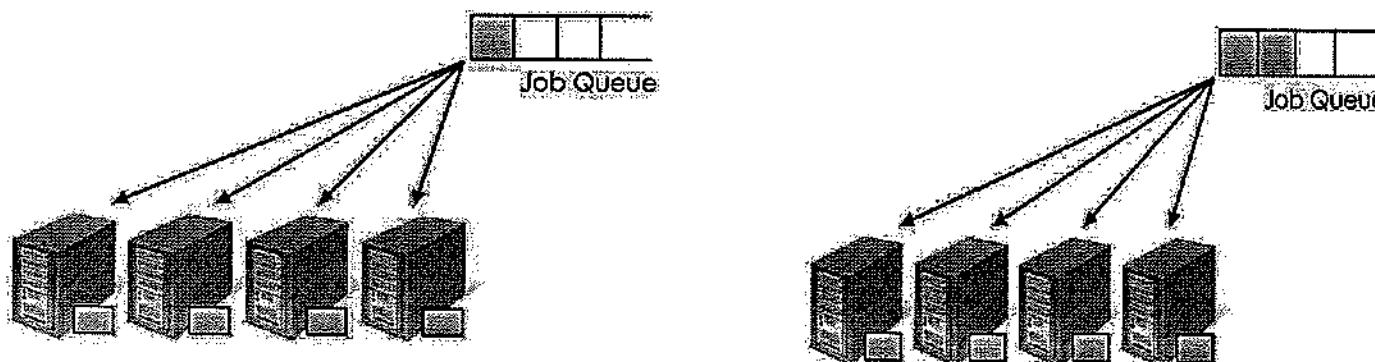
\* *It is possible to change the job weight, but we cannot preempt a running job.*

Ex: `hadoop job -set-priority job_ID HIGH`



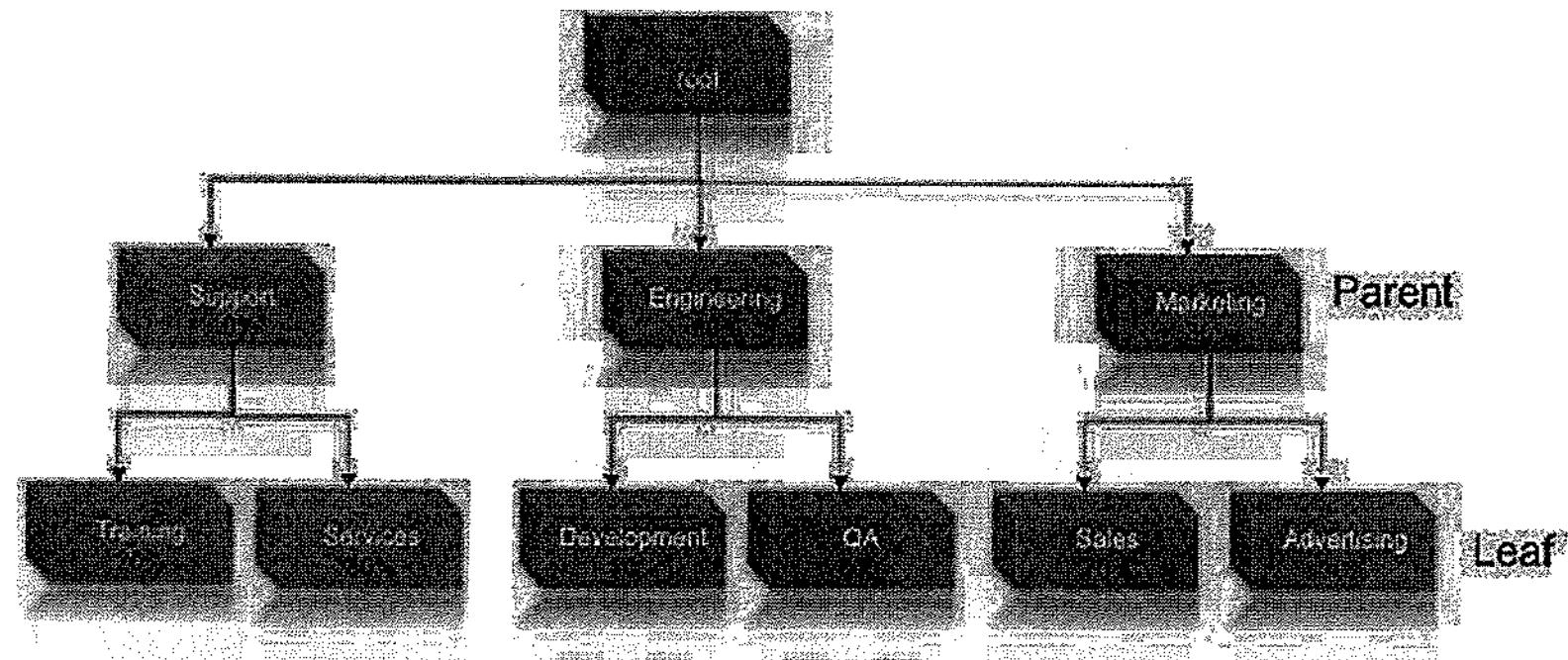
## Fair Scheduler

- JT/RM divides the resources equally among the queues & jobs
- Very similar to the time slicing model followed by the operating system schedulers
- Each user is given a fair share of resources in the queue
- JT/RM Kills the running resources once the *pre-emption* time period (*default 15s*) is over



## Capacity Scheduler

Use when we have *non-homogeneous* hardware (different configuration of Nodes) in the Cluster. Each organization is set up with a dedicated queue that is configured to use a given fraction of the Cluster capacity. Queues may be further divided in hierarchical fashion, allowing each organization to share its Cluster allowance between different groups of users within the organization. Within a queue, applications are scheduled using FIFO scheduling.



## YARN Resource Allocation Models

- ***Steady Fair Share***

Fair Share is simply dividing the YARN Cluster resources (memory and vcores) across multiple queues as per the allocation.

**Example:**

```
<schedulingPolicy>fair</schedulingPolicy>
<minResources>20480 mb, 24 vcores</minResources>
<maxResources>2048000 mb, 2000 vcores</maxResources>
```

*minResources* means the soft limit of resources guaranteed to the particular queue.

*maxResources* means the hard limit of resources that can be used at max if resources are free.

- ***Instantaneous Fair Share***

In Instantaneous Fair Share, Resource limit for each queue will be calculated on the fly based on number of active queues at that particular moment. If all the queues are active, it just acts like Steady Fair Share.



## YARN Resource Allocation Models (Cont'd)

- ***Dominant Resource Fairness (DRF)***

In Steady & Instantaneous Fair schedulers, scheduling fairness decisions will happen by considering only memory availability. DRF is configured in such a way to schedule by considering both memory and vcores. This will be very helpful when you are running Memory demand jobs (ex: MR/Spark Jobs) and CPU demand jobs (ex: Machine Learning Jobs) separately in the same Cluster.

`<schedulingPolicy>drf</schedulingPolicy>`

- ***Dynamic Resource Pools (Only in Cloudera, for both YARN & Impala)***

A Dynamic Resource Pool is a named configuration of resources and a policy for scheduling the resources among YARN applications and Impala queries running in the pool. Dynamic resource pools allow you to schedule and allocate resources to YARN applications and Impala queries based on a user's access to specific pools and the resources available to those pools. If a pool's allocation is not in use it can be given to other pools. Otherwise, a pool receives a share of resources in accordance with the pool's weight. Dynamic resource pools have ACLs that restrict who can submit work to and administer them.



## Performance Tuning of a MapReduce Job

- *Number of Mappers*

How long are your mappers running for? If they are only running for a few seconds on average, better to have fewer mappers and make them all run longer. As a thumb rule, each mapper should run for 1 minute or so.

- *Number of Reducers*

Check that you are using more than a single reducer. As a thumb rule, Reducer tasks should run for 5 minutes or so and produce at least one full block of data.

- *Usage of Combiners*

Check whether your job can take advantage of a combiner to reduce the amount of data (map output) passing through the shuffle.

- *Intermediate compression*

Job execution time can almost always benefit by enabling map output compression

- *Speculative Execution*

- *Distributed Cache*





## Performance Tuning of YARN Job

- *Compression Levels*
  - Input Data
  - Intermediate Map Output
    - `mapreduce.map.output.compress`
    - `mapreduce.map.output.compress.codec`
  - Reducer Output
    - `mapreduce.output.fileoutputformat.compress`
    - `mapreduce.output.fileoutputformat.compress.type`
    - `mapreduce.output.fileoutputformat.compress.codec`

LZO / Snappy Compression techniques are well suited for hadoop

- *JVM Reuse policy*
  - `mapreduce.job.jvm.numtasks`

By default, 1 (No reuse). Can be set to -1 to schedule unlimited number of tasks on particular JVM. This will be helpful if you are running large number of short running tasks.



## Performance Tuning of YARN Job (Cont'd)

- *Split Size*

- `mapreduce.input.fileinputformat.split.minsize` – 134217728 (128 MB)

Better to run small number of longer running map tasks than large number of very short running map tasks. Need to consider spills if bigger split size than block size.

- *Map-Side Spills*

- `mapreduce.task.io.sort.mb` – 100 MB
    - amount of buffer memory to use while sorting files
  - `mapreduce.map.sort.spill.percent` – 80%

An easy way to detect if the Map phase is performing additional spills is to look at “Map output records” and “Spilled Records” counter of JT/RM page for the job immediately after the Map phase is completed. If the number of spilled records are more than Map output records, then additional spilling is happening.

## Performance Tuning of YARN Job (Cont'd)

- *Reduce-Side Spills*

- `mapreduce.reduce.shuffle.input.buffer.percent` – 80%

80% of Heap Memory will be used to spill. You can set to 0 to use complete 100% of JVM

- *Copy/Shuffle Phase Tuning*

- `mapreduce.reduce.shuffle.parallelcopies` – 5

The maximum number of parallel map-output copier threads in `mapred-site.xml` is set to 5 by default. This could be a limiting factor for the throughput of copy operation.

- *Blacklisting NodeManagers*

- `mapreduce.job.maxtaskfailures.per.tracker` – 3

The `mapreduce.job.maxtaskfailures.per.tracker` configuration parameter of `mapred-site.xml` which governs when a NodeManager gets blacklisted is set to 3 by default. This means that if any of the NodeManagers have 3 or more task failures then those NodeManagers will be blacklisted and no further scheduling of tasks will happen on them.



## Performance Tuning of YARN Job (Cont'd)

- ***YARN configuration***

The minimum number of vcores should be 1. When additional vcores are required, adding 1 at a time should result in the most efficient allocation. Set the maximum number of vcore reservations for a container to ensure that no single task consumes all available resources.

*yarn.scheduler.minimum-allocation-vcores=1*

*yarn.scheduler.maximum-allocation-vcores=8*

*yarn.scheduler.increment-allocation-vcores=1*

The minimum and maximum reservations for memory. The increment should be the smallest amount that can impact performance. Here, the minimum is approximately 1 GB, the maximum is approximately 8 GB, and the increment is 512 MB.

*yarn.scheduler.minimum-allocation-mb=1024*

*yarn.scheduler.maximum-allocation-mb=8192*

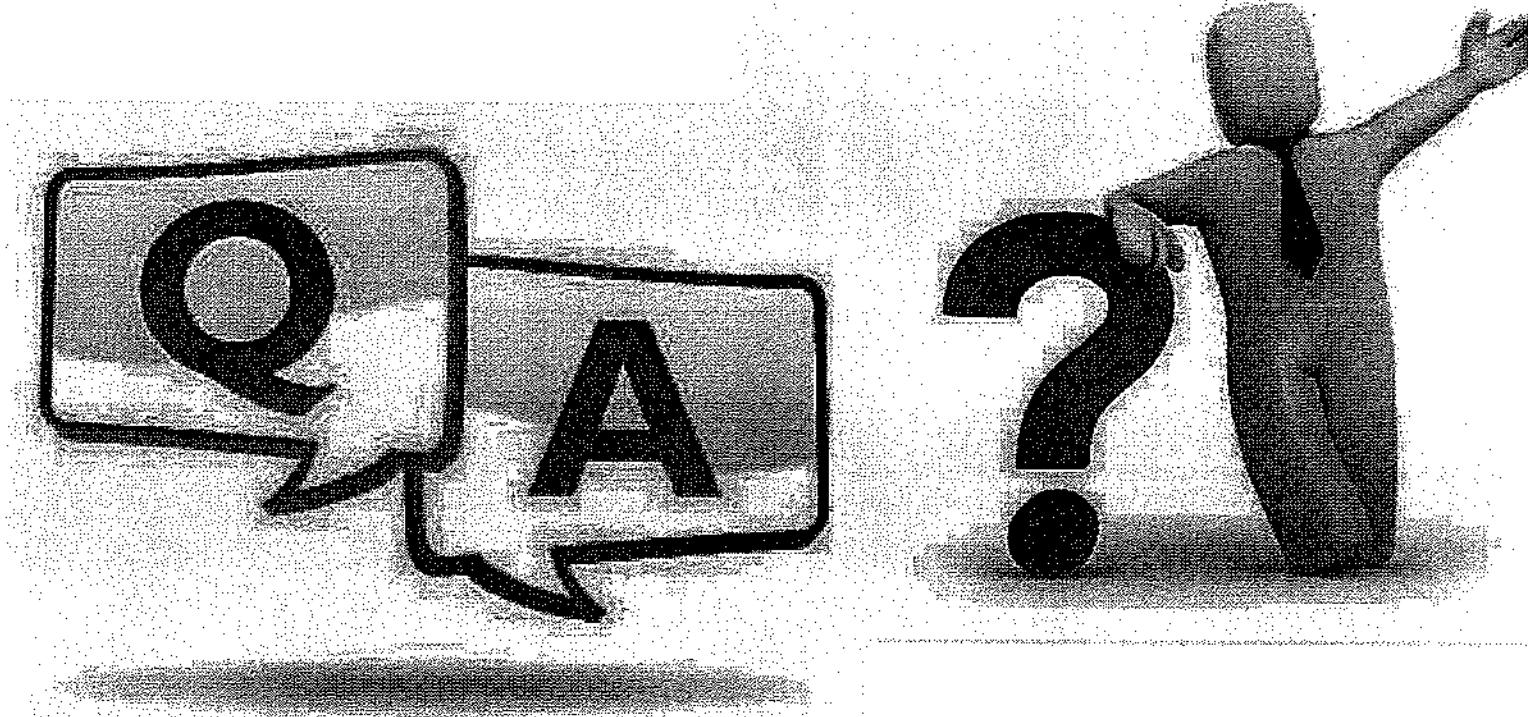
*yarn.scheduler.increment-allocation-mb=512*



## **Hadoop Admin Responsibilities**

- Cluster planning, setup and deploy new hardware and software required for Hadoop
- Responsible for implementation and ongoing administration of Hadoop infrastructure
- Managing Linux users, setting up Kerberos principals and providing access to the required Hadoop eco-system tools for the new users
- Commissioning & decommissioning of Cluster Nodes to & from the Cluster
- Managing alerts from Cluster monitoring tools like Ganglia, Nagios and CM / Ambari / MCS.
- Performance tuning of Hadoop Clusters and Hadoop jobs
- Manage and review Hadoop Log files
- Disk space management and monitoring
- HDFS support and maintenance
- Working together with infrastructure, network, database, application and business intelligence teams to guarantee high data quality and availability
- Working together with application teams to install operating system and Hadoop updates, patches, version upgrades when required
- Escalate the issue to the Vendor if it can't be resolved internally







thanks

