

Lecture 04

Hadoop Distributed File System (HDFS)

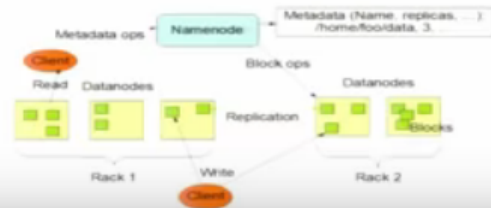
Hadoop distributed file system, HDFS.

Refer Slide Time :(0: 18)

Preface

Content of this Lecture:

- In this lecture, we will discuss design goals of HDFS, the read/write process to HDFS, the main configuration tuning parameters to control HDFS performance and robustness.



Preface content of this lecture; in this lecture, we will cover the goals of HDFS, read/write process in HDFS, configurations tuning parameters to control HDFS, performance and robustness.

Refer Slide Time :(0: 37)

Introduction

- **Hadoop provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce paradigm.**
- **An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data.**
- **A Hadoop cluster scales computation capacity, storage capacity and IO bandwidth by simply adding commodity servers. Hadoop clusters at Yahoo! span 25,000 servers, and store 25 petabytes of application data, with the largest cluster being 3500 servers. One hundred other organizations worldwide report using Hadoop.**

Introduction, Hadoop provides distributed file system, as a framework for the analysis and performance of very large, datasets computations using Map Reduce paradigm. Important characteristics of Hadoop is the partitioning of data and computation across hundreds and thousands of the nodes of a cluster and executing these computations, in parallel, close to their data, Hadoop cluster scales computations capacity, storage capacity and i/o bandwidth by simply adding commodity servers, Hadoop clusters at Yahoo! spans 25,000, servers and store 25 pet bytes of application data, with the largest cluster being 3,500, servers 100 other organization worldwide report using Hadoop.

Refer Slide Time :(1: 52)

Introduction

- Hadoop is an **Apache project**; all components are available via the Apache open source license.
- **Yahoo!** has developed and contributed to **80%** of the core of Hadoop (**HDFS and MapReduce**).
- **HBase** was originally developed at **Powerset**, now a **department at Microsoft**.
- **Hive** was originated and developed at **Facebook**.
- **Pig, ZooKeeper, and Chukwa** were originated and developed at **Yahoo!**
- **Avro** was originated at **Yahoo!** and is being co-developed with **Cloudera**.

With this introduction let us, see the picture of our overall viewpoint of Hadoop. So, Hadoop is an Apache project; all the components are available via Apache open source license. And Yahoo! has developed and contributed 80 percent of the core that is, core of Hadoop that is, HDFS and Map Reduce. HBase was originally developed at Power set, now Microsoft. Hive was originated and developed Face book. Pig, zookeeper, Chukwa were originated and developed at Yahoo!

Refer Slide Time :(2: 38)

Hadoop Project Components

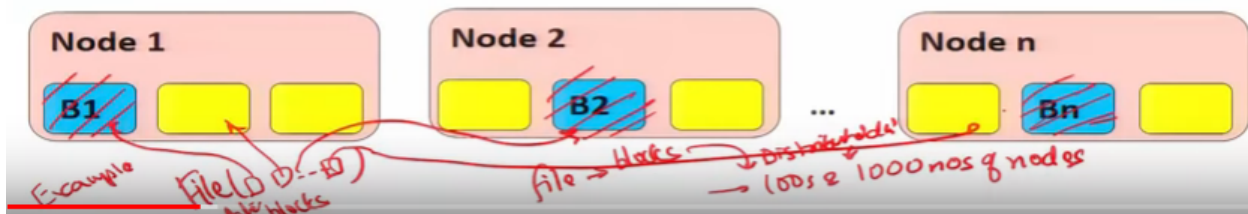
HDFS	Distributed file system
MapReduce	Distributed computation framework
HBase	Column-oriented table service
Pig	Dataflow language and parallel execution framework
Hive	Data warehouse infrastructure
ZooKeeper	Distributed coordination service
Chukwa	System for collecting management data
Avro	Data serialization system

So, some of the Yahoo! Project components and the first and foremost is the HDFS: that is Hadoop distributed file system and the other important component of Hadoop project is, the Map Reduce that is distributed parallel computation framework.

Refer Slide Time :(2: 57)

HDFS Design Concepts

- **Scalable distributed filesystem:** So essentially, as you add disks you get scalable performance. And as you add more, you're adding a lot of disks, and that scales out the performance.
- **Distributed data on local disks on several nodes.**
- **Low cost commodity hardware:** A lot of performance out of it because you're aggregating performance.



Now, let us see the Hadoop distributed file system: some of the design concepts and then we will go in more detail of HDFS, design. Now, the first important thing is about a scalable, distributed file system: that means, we can add more disk and we can get a scalable performance that is one of the major design, concepts of HDFS, in realizing the scalable distributed file system. Now, as you add more you are adding

a lot of disk and, this will automatically scales out the performance, as far as the design, goal of HDFS is concerned. Why this is required is because, if the data, set is big and very large, cannot fit in, to one computer system. So, hundreds and thousands of computer systems are used, to store that file. So, hence the data, of a file is now, divided into the form of a blocks and distributed, onto this large-scale infrastructure. So, that means a distributed data, on the local disk is now, stored on several nodes, this particular method, will insure the low cost commodity hardware usage, to store the amount of information, by distributing it across all these multiple nodes, which comprises of low-cost commodity hardware. The drawback is that, some of the nodes may get failure: that we will see that, it is also included as per the design, goal of HDFS and the low cost commodity hardware, is to be used in this particular manner, a lot of performance out of it, is achieved because, we are aggregating the performance, of hundreds and thousands of such commodity low cost hardware. So, in this particular diagram we see that, we assume and number of nodes let us say node 1, node 2, on so on, node n these nodes are, in the range of hundreds and thousands of the nodes and if a file is given, file is broken into the, to the blocks and these blocks are now, having a data, which will be distributed, on this particular kind of setup. So, in this particular example, we can see that, a file is there and that file is, divided into the blocks, file data is block, in divided into the data, blocks and each block, is stored across different nodes. So, we can see here, all the blue colored nodes, blue color blocks of these nodes are storing a file data. So, hence the file data is now distributed, onto this local disk in HDFS.

Refer Slide Time :(7: 02)

HDFS Design Goals

- **Hundreds/Thousands of nodes and disks:**
 - It means there's a higher probability of hardware failure. So the design needs to handle node/disk failures.
- **Portability across heterogeneous hardware/software:**
 - Implementation across lots of different kinds of hardware and software.
- **Handle large data sets:**
 - Need to handle terabytes to petabytes.
- **Enable processing with high throughput**

So, hundreds and thousands of the nodes are available and their disk is being used for storing. Now, these comprises of the commodity hardware, so they are prone to the hardware failure and as I told that, that this design. So, they are prone to the hardware failure, so the design needs to handle, the node failures. In this particular case, so HDFS design goal, is to handle, the node failures also. So, another aspect is about the portability across heterogeneous Hardware, why because? There are hundreds and thousands of community hardware machines, they may be having different operating system and the software running, so hence, this heterogeneity also requires, the portability support, in this particular case. That is also one

of the HDFS design goals, another important design goal of HDFS is to handle, the large data sets. So, the data sets so the file size, ranging from terabyte to the pet bytes that is huge, file or huge dataset is also now, being able to stored here in HDFS file system so it provides a support of, the handling the large datasets also enable the processing with the high throughput. So that means how this is all ensured the processing with the high throughput? That we will see and it has kept as one of the important design goals of HDFS.

Refer Slide Time :(8: 44)

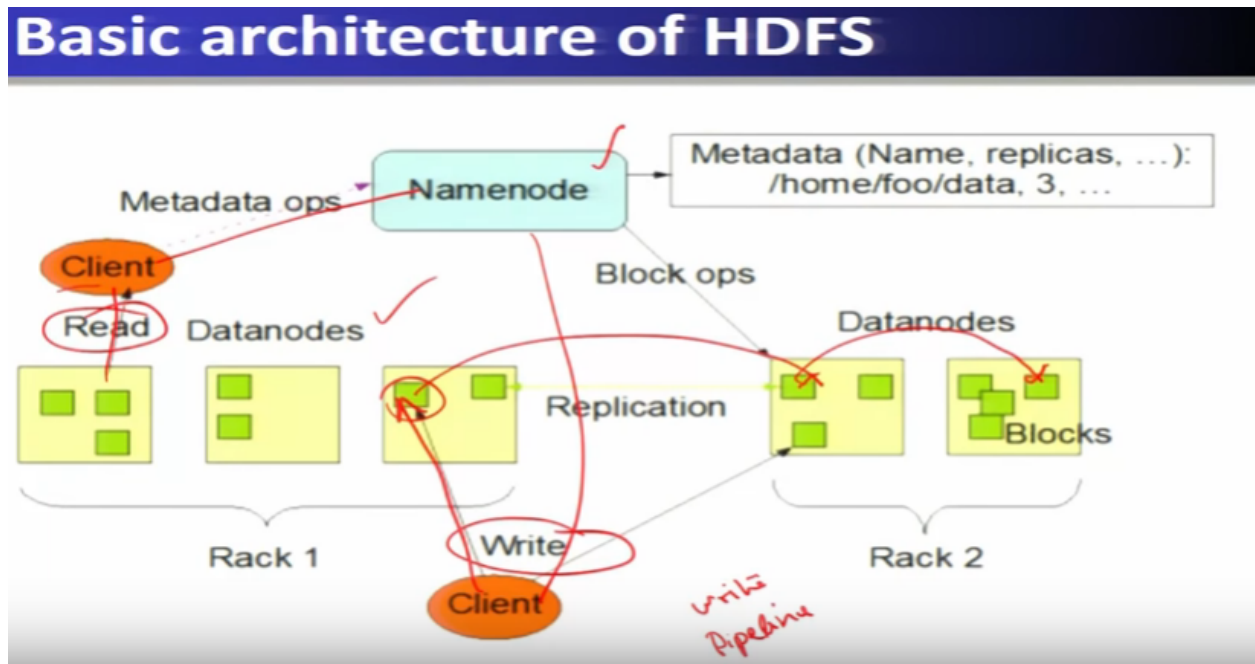
Techniques to meet HDFS design goals

- **Simplified coherency model:**
 - The idea is to write once and then read many times. And that simplifies the number of operations required to commit the write.
- **Data replication:**
 - Helps to handle hardware failures.
 - Try to spread the data, same piece of data on different nodes. *- Replicate*
- **Move computation close to the data:**
 - So you're not moving data around. That improves your performance and throughput.
- **Relax POSIX requirements to increase the throughput.**

Now let us, understand the techniques to meet these design goals, the first of this technique is called a, 'Simplified Core' and model that is, which is nothing but, right once and read many times, this will simplify the number of operations, hence since we are going to write once and read many times. So, most of the design will be based on that coherent model. Another technique which will meet these design goals, which we have seen in the previous slide, is about the data application. Now, since there is a, possibility of hardware failures or a failing of these nodes, which are of commodity Hardware therefore, the data blocks, which are store in this particular HDFS file system, is replicated at more than one nodes and hence the data application, is the technique, which will be there to handle the hardware failures. So, by data application means the data blocks, will be spirited, across different nodes and add more than one times, these replications are the same piece of data, is now available on different nodes using replication. So, it is not that only one copy of a data is stored but, the replication factor or replication we'll tell how many, different same piece of data, is stored on how many nodes? So, even if that node is fail or Iraq is failed, even then there is a possibility that, the data is available and it will overcome from such failures that is done through the hardware data application. Another important technique which basically ensures, the high performance throughput, is that we moved, to the computation rather computation will be moving close to wherever the data is there hence, we are not moving the data around and this will improve the performance and the throughput of the system. Another technique which is used to meet the HDFS design goal is that, we will relax some of the POSIX requirements, we increase the throughput for example, when I write by the client, then the write operation, will keep on doing the cache, at the clients

end. So, this basically, is the relaxation of the POSIX and this will increase the throughput that we will see later on in this particular part of the discussion.

Refer Slide Time :(11: 37)



So, this is the basic architecture of HDFS, which comprises of a single name node and multiple data nodes and which supports the two operations which are write and read, by the clients, so and whenever this, particular client want to do a read operation, a write operation then it has to contact to the name node, try to find out the data, nodes where these particular blocks of a file, need to be written and out of them one, the client will write to the, closest of these data blocks and that particular data block in turn will replicate, through the other data nodes and that particular data node in turn will replicate to the other data node, which is the, which is in the same rack. So, this way, the entire operation: that means this writing on a one block and then replicating at other data nodes, will happen at the same point of time hence, the right is done in, a pipeline mode. This will increase the throughput, of this right operation, similarly whenever a read operation, is required by the client, then this client will contact to the name node, try to find out the blocks or data nodes, where that blocks are stored and out of them, the client will prefer to read, from the data block which is very close to that particular client. In this way the throughput is increased so, therefore a name node and data node together, will which constitutes the architecture of HDFS, is able to support this large-scale data storage and also, ensure the computations at that, place with a high throughput.

Refer Slide Time :(13: 50)

HDFS Architecture: Key Components

- **Single NameNode:** A master server that manages the file system namespace and basically regulates access to these files from clients, and it also keeps track of where the data is on the DataNodes and where the blocks are distributed essentially. *- metadata -*
- **Multiple DataNodes:** Typically one per node in a cluster. So you're basically using storage which is local.
- **Basic Functions:**
 - Manage the storage on the DataNode.
 - Read and write requests on the clients
 - Block creation, deletion, and replication is all based on instructions from the NameNode.

So, let us see the HDFS architecture, again and describe its key components. So, a single name node that we have seen, is nothing but, a master server that manages the file system name, namespace and basically regulates, access to these files from the client and also, keep track of where the data is on the data node and where the blocks, are distributed essentially. So, single name node will store the Meta, data that means the information about the data, where it is, stored on the data nodes, is maintained by the name space. Now, as far as the data which is actually stored on the data multiple data nodes. So, that means one typically, one per node, in a cluster, is there that is maintained by the name node information and which is used to store the, the data locally. So, the basic functions here, in this key components are that of HDFS is to manage, the storage on the data nodes, where the actual data, is managed or is told and read and write requests, are being initiated by the client, into the HDFS support in the HDFS architecture, similarly for the block creation, deletion and the replication is all based on the instructions, from the name node. So, name node is, basically managing the entire operations of the data, placement data axis, in terms of block creation, deletion and replication.

Refer Slide Time :(15: 44)

Original HDFS Design

- Single NameNode
- Multiple DataNodes
 - Manage storage- blocks of data
 - Serving read/write requests from clients
 - Block creation, deletion, replication

So, we have seen that, in the original HDFS design there is single name node and a multiple data nodes and these data nodes, will manage the storage that is nothing but a blocks of data and these data nodes are serving, for the read and write request from the initiated by the client and also these data nodes, will perform the block creation, deletion and replication.

Refer Slide Time :(16: 07)

HDFS in Hadoop 2.0

- **HDFS Federation:** Basically what we are doing is trying to have multiple data nodes, and multiple name nodes. So that we can increase the name space data. So, if you recall from the first design you have essentially a single node handling all the namespace responsibilities. And you can imagine as you start having thousands of nodes that they'll not scale, and if you have billions of files, you will have scalability issues. So to address that, the federation aspect was brought in. That also brings performance improvements.
- **Benefits:**
 - Increase namespace scalability ✓
 - Performance ✓
 - Isolation ✓

Now let us see, what is new in Hadoop version 2.0? So, HDFS, in a version Hadoop 2.0 or HDFS 2.0, uses the HDFS Federation that means that, it is not a single namespace but it is a Federation, as that is called, 'HDFS' name node Federation. So, so this particular Federation will now, have multiple data nodes and multiple name nodes, are there and this will increase, the reliability of the name node, in this case of Federation. So, it is not one name node, but it is, n number of n in name nodes and this particular method is called the, 'HDFS Federation'. The benefits is to increase the namespace scalability, earlier there was one name is space now it has, a Federation of name a space so, obviously the scalability is increased and also, the performance is increased and also, the isolation, performance is increased, why

because? Now the, the, the nearest namespace is used to serve, the clients requests and isolation means if let us say a high, requirement high a large amount of resource requirement is there for a particular application, it is not going to affect, the entire a single namespace, why because? It is a Federation of name is space other, applications is not going to be affected by a very high, requirement for a particular application that is called, 'Isolation'.

Refer Slide Time :(18: 00)

HDFS in Hadoop 2

- How its done
 - Multiple Namenode servers ✓
 - Multiple namespaces ✓
 - Data is now stored in Block pools
-
- So there is a pool associated with each namenode or namespace.
 - And these pools are essentially spread out over all the data nodes.

Now, in HDFS version two, are in Hadoop version two how, this all is done let us go and discuss. Now, here as we have mentioned that, instead of one name node here now we have, multiple name mode servers and they are managing, the namespace hence they becomes a multiple namespace and the data, is now stored in the form of a block pools. So, now this block pool is also, going to be managed across these data nodes, on the nodes of the cluster machines. So, it's not only one node, but several nodes are involved and they will be storing the block pools. So, there is a pool associated with each name node or a namespace and these pools are essentially, spread out over all the data nodes.

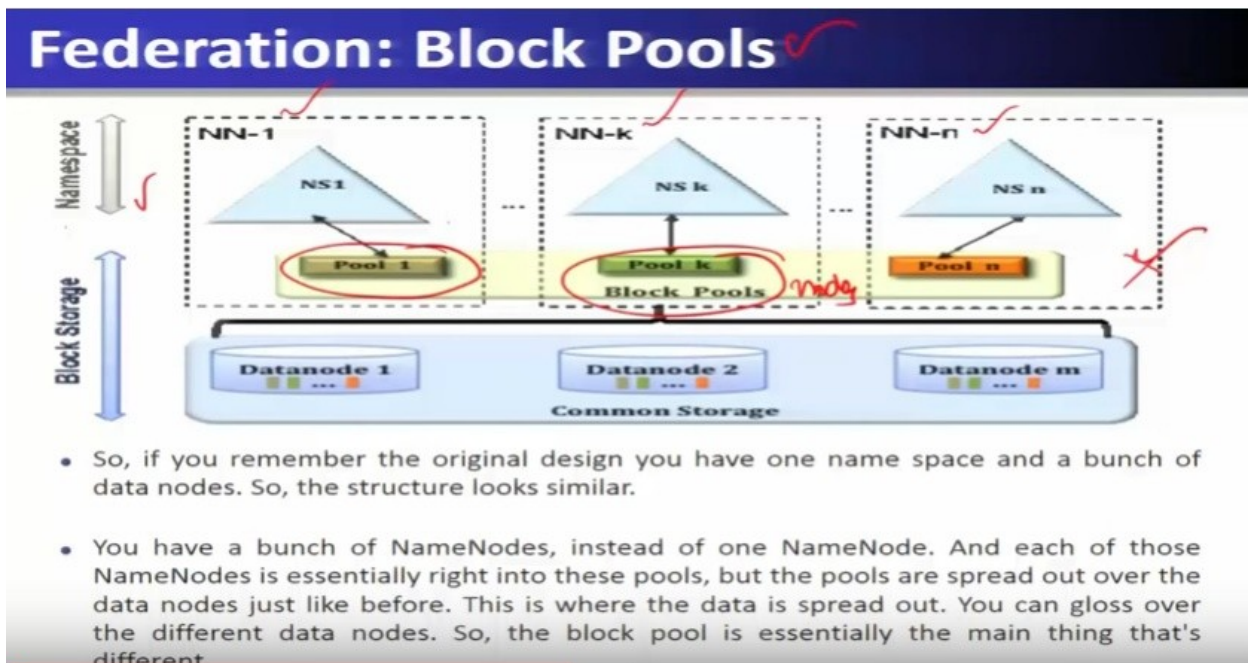
Refer Slide Time :(19: 01)

HDFS in Hadoop 2

- High Availability-
Redundant NameNodes
- Heterogeneous Storage
and Archival Storage
 - ARCHIVE, DISK, SSD, RAM_DISK

That we will see, in the further picture.

Refer Slide Time : (19: 04)



So, here you can see in this particular diagram that the namespace it has multiple namespace, name a space one, name is space 2 and so on up to namespace n. Let us assume that it has, multiple namespaces and each name is space, is having a block pool. So, these are called, 'Blocks Pools' and these block pools are, stored on the nodes. So, they are on different nodes, just like a cluster machine so, each block pool is stored on a different node. So, different nodes are there and this is going to manage, the multiple namespace and this is called the, 'Federation' of the block pools. So, hence now it is not a single point of

failure, even if one or more name node, namespace or a name node fails, it is not going to affect, anything and it also increases the, the performance reliability and throughput and also, performs also provides you the isolation. So, if you remember the original design you have only one namespace and a bunch of data nodes, so the structure looks, like similar but, internally it is managed as the Federation. So, you have a bunch of named nodes now, instead of one named node and each of these named nodes is essentially write to these pools. But, the pools are spread out over the data nodes just like before, this is where the data is spread out and you can grass over the different data nodes. So, the block pool is essentially the main thing that's different in Hadoop, version or HDFS version two.

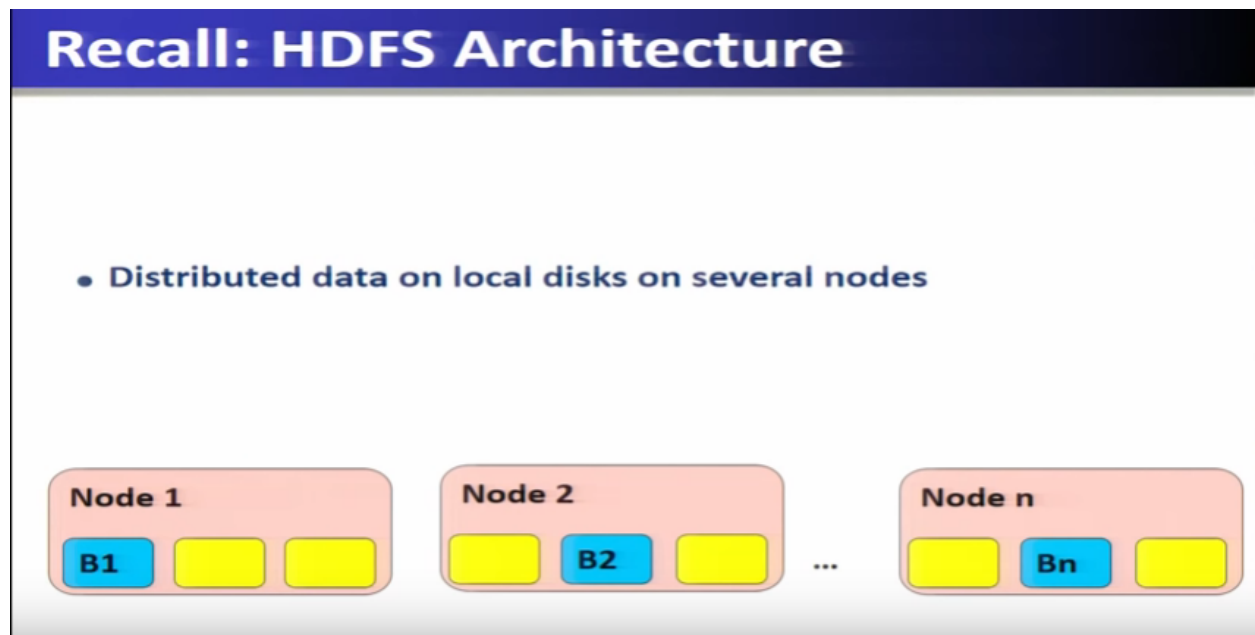
Refer Slide Time :(20:45)

HDFS Performance Measures

- Determine the number of blocks for a given file size, ↓ Replication - 3
- Key HDFS and system components that are affected by the block size. - 64 MB to 128 MB
↑ → ↑
- An impact of using a lot of small files on HDFS and system 4

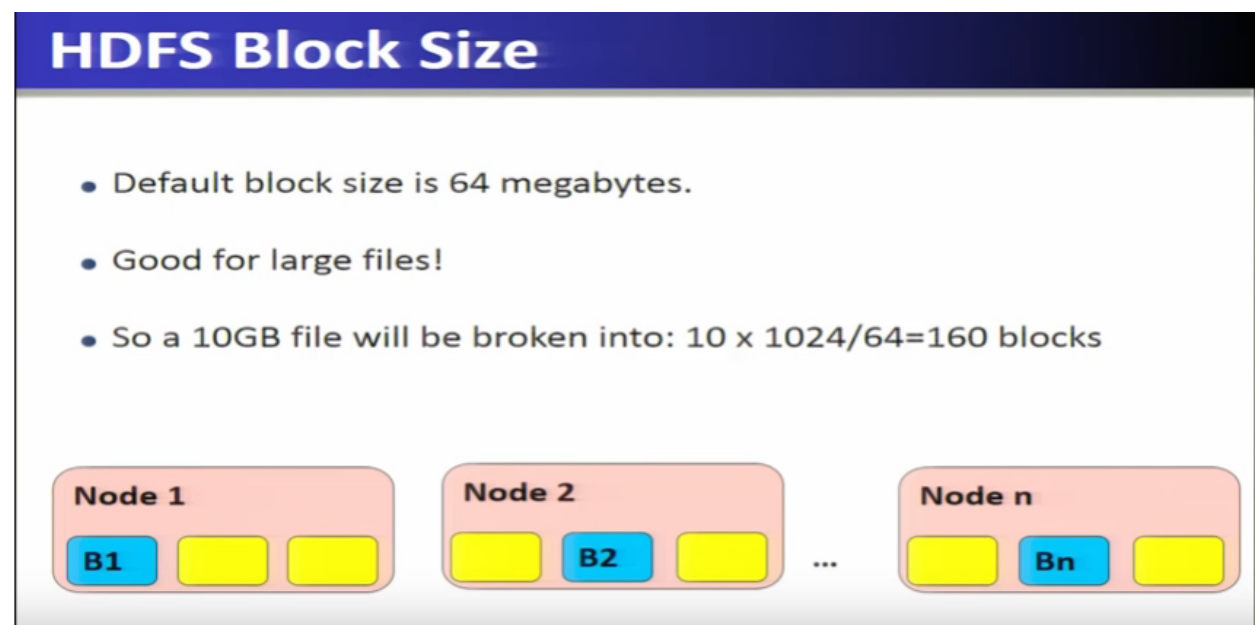
So, HDFS performance measures, if we see that, here we see that, determine the number of blocks, for a given file, for a given file size. And the, the key HDFS and the system components are affected by the block size and impact of ,using a lot of small files on, HDFS and HDFS system. So, these are some of the performance, measures that we are going to, tune the parameters and measure the performance. Let us summarize it again, all these and different, tuning parameter for performance. And so, basically the number of, how many number of blocks for a given file size? And this is required, to be known, in so basically, we will see there is a performance measure or, or basically there is a tradeoff, in the number of blocks, to be replicated. The another key component is, about the size of the block. So, here the block size, which varies from 64 MB to 128 MB. So, if the block size is 64 then, what is the performance and if we increase the block size, then what will be the performance, similarly the number of blocks, that means, how many this is the replication, if the replication factor is three that means, every block is replicated on three different nodes, for a given file size. So, if the replication is 1, then obviously we are, saving lot of space. But, performance we are going to sacrifice, so there is a trade-off between this. And another important parameter, for HDFS performance is about, the number of small files, which are there on, the HDFS. So, if there are lot of small files, which are there in HDFS, the performance goes down, we will see how and how this particular problem is to be overcome, in the further slides.

Refer Slide Time :(23:08)



So, let us see that, recall again the HDFS architecture, where the data is distributed, on the local disk, on several nodes. So here, in this particular picture, we have shown, several nodes where data is divided and that is called, 'Distributed Data on Different Local Disk', it is stored.

Refer Slide Time :(23:31)



Now, the data is stored in the terms of block and the block size is going to matter much, in the performance, the default block size is 64 megabytes. And this is good for a large file. So, if there is a file size of 10 GB, then this particular file is broken into, 160 blocks of size 64 megabytes. So, 160 blocks

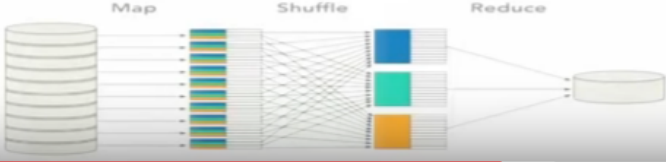
we have to just store, in a distributed manner, on several nodes and therefore, this particular block size is going to matter much. So, if we increase the number of or the size of the block, then obviously, it will be less than 160 blocks. So, if the number of blocks are more hence, the parallel operations is more possible and that we are going to see about, what is the, what is the effect of keeping the small block size, of 64 or a more than 64.

Refer Slide Time :(24:34)

Importance of No. of Blocks in a file

NameNode memory usage: Every block that you create basically every file could be a lot of blocks as we saw in the previous case, 160 blocks. And if you have millions of files that's millions of objects essentially. And for each object, it uses a bit of memory on the NameNode, so that is a direct effect of the number of blocks. But if you have replication, then you have 3 times the number of blocks.

Number of map tasks: Number of maps typically depends on the number of blocks being processed.



So, the importance of the number of blocks in a file. So, if the number of blocks are more, than the amount of memory which is used in the name node, will be more in that case. So, for example, here every block that, you create basically every file, could be a lot of blocks we saw in the previous case, 160 blocks. and if you have a million of files and this, that millions of objects, essentially is required to basically store that, amount of space in the name node to manage it and it becomes, several times bigger. If let us say, the number of blocks are more. And the files are more. So, we will see this kind of importance, of the number of blocks, so it is going to affect, into the size, into the name node. And it measures, how much memory is going to be used, in the name node to manage that, many number of blocks of a file. Now, the number of map tasks, also is going to matter much, for example, if the file is divided into 160 blocks, so at least 160 different, map functions are required to be executed, to cover entire data set operations or computations. So hence, if the number of blocks are more, not only it is going to take more space, in the name node, but also more number of map, functions also required to be executed.

Refer Slide Time :(26:16)

Large No. of small files: Impact on Name node

- **Memory usage:** Typically, the usage is around 150 bytes per object. Now, if you have a billion objects, that's going to be like 300GB of memory.
- **Network load:** Number of checks with datanodes proportional to number of blocks

Hence, there has to be a tradeoff. Similarly, if there is a large number of a small files: this will impact on the name node, why because a lot of memory is required, to store the information of this number of small files. Hence, the network load is going to increase, in this particular case.

Refer Slide Time :(26:37)

Large No. of small files: Performance Impact

- **Number of map tasks:** Suppose we have 10GB of data to process and you have them all in lots of 32k file sizes? Then we will end up with 327680 map tasks.
- Huge list of tasks that are queued.
- The other impact of this is the map tasks, each time they spin up and spin down, there's a latency involved with that because you are starting up Java processes and stopping them.
- Inefficient disk I/O with small sizes

So, if there is a large number of a small file: there is a performance issue or a problem of a performance. So, suppose you have 10 GB of data, to process and you have all in a lots of 32 KB of a file size? Then we are end up with, so many number of map tasks. So, huge list of tasks are now queued up and the performance will go down, why because, when they spin up and spin down ,there is a latency involved

and because, you are starting up the Java and stopping them and also, it is inefficient due to the disk i/o, with the small sizes.

Refer Slide Time :(27:19)

HDFS optimized for large files

- Lots of small files is bad!
- **Solution:**
 - Merge/Concatenate files ✓
 - Sequence files ✓
 - HBase, HIVE configuration ✓
 - CombineFileInputFormat ✓

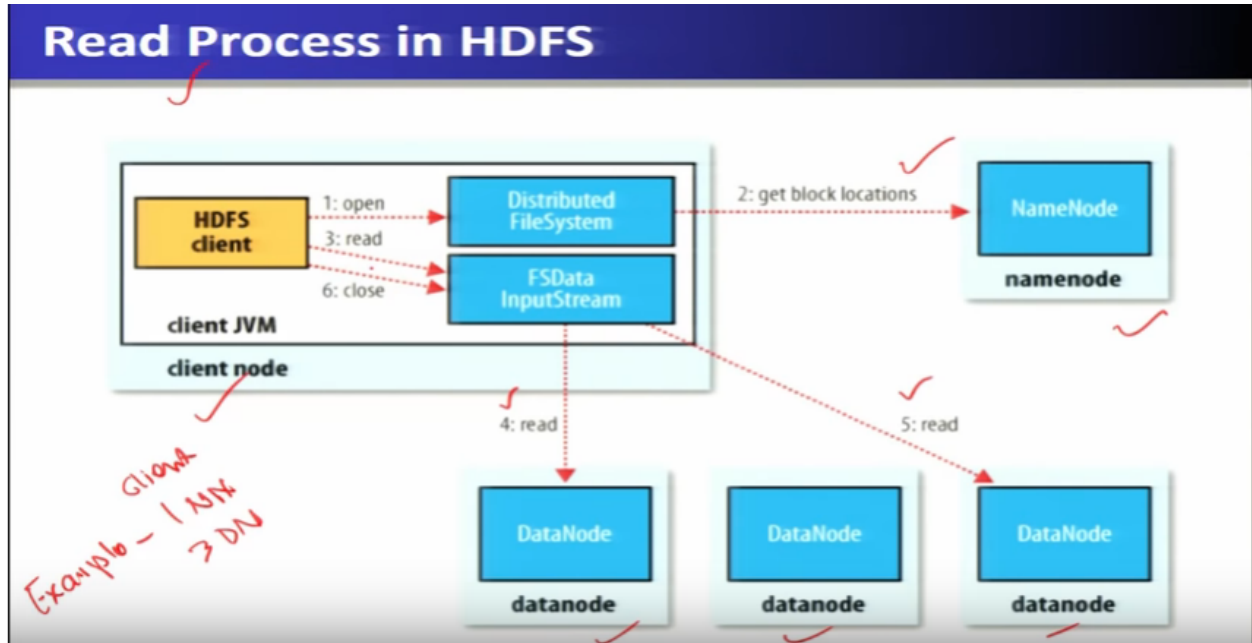
So, HDFS is therefore optimized for a large file size. So, lot of small files is bad and the solution, to this particular problem is to merge and concatenate different files are, there is a operation which is called, 'Sequence Files', several files are Mouse together, in a sequence and that is called a, 'Sequence File'. And which is treated as one big file instead of keeping, many number of small files, another solution, to the small number of lots of small number of files is, using the HBase and hive configuration, for this particular large number of small files. They will be used to optimize this particular issue. And also there is also, another solution is to combine the input file, input format, file input format.

Refer Slide Time :(28:20)

Read/Write Processes in HDFS

Now, let us see, in more detail about read and write processes, which is there in HDFS, how it is being supported.

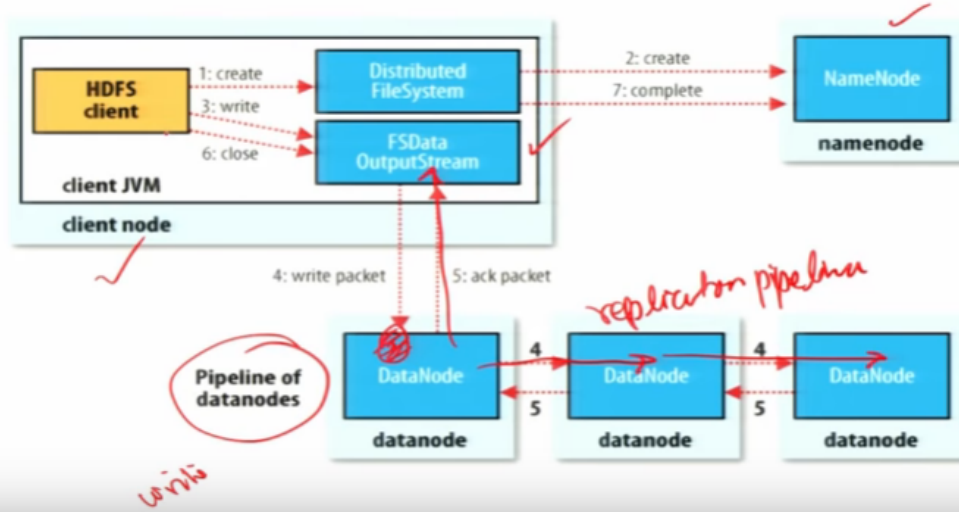
Refer Slide Time :(28:26)



Now, the read process in HDFS, if we see that, first of all we have to identify that there is a name node. And this is the client. And there are several data nodes, in this example, we are having one name node and there are three data nodes and there is a client, which will perform the read operation. So, so the HDFS client, will then request to read a particular file, this is the read operation and this particular request will go to the name node, to know the, the blocks where that, read operation is to be executed and the data is to be given back, to the to the client. So, it will send the request to the name node and then, name node will, give back this information, back to this particular client end of HDFS and from there, it will have two options, whether to read from the block number four or to read from the block number five. Then it will try to read the, the one which is the closest one and this particular data is, given back to the client.

Refer Slide Time :(29:56)

Write Process in HDFS



Now, then let us see the write operation, which is initiated again by the client. So, whenever there is a client wants to do a write operation and this particular write operation is now, going to be requesting, the name node to find out the, the data node which can, be used to store, the, the clients data. And after getting this information back, this particular right operation is being performed on this particular data node, which is, which is the closest one and that, particular data node has to, do this replication. If let us say, replication factor is 3 then, it will do this in the form of a pipeline. So, the client will write down or will write the data, on a particular data node and that data node, in turn will carry out, the pipeline for the replication, this is called a, 'Replication Pipeline'. And once the replication is over, then it will send the acknowledgment and the right operation is completed, in this particular process.

Refer Slide Time :(31:18)

HDFS Tuning Parameters

Now, let us see in more detail about, HDFS tuning parameters.

Refer Slide Time :(31:24)

Overview

- Tuning parameters
- Specifically DFS Block size
- NameNode, DataNode system/dfs parameters.

So, HDFS tuning parameters, we are going to see, especially the DFS block size, from that viewpoint and also we will see the name node, data node and all these different tuning parameters.

Refer Slide Time :(31:37)

HDFS XML configuration files

- Tuning environment typically in HDFS XML configuration files, for example, in the hdfs-site.xml.
- This is more for system administrators of Hadoop clusters, but it's good to know what changes affect impact the performance, and especially if your trying things out on your own there some important parameters to keep in mind.
- Commercial vendors have GUI based management console

Now, as far as tuning parameter is concerned in HDFS, there is a file XML configuration file. And for example, HDFS site dot XML file is there, where this particular environment or configuration parameter can be set. In some of the cases like, cloud era, supports, automatic, GUI for these configurations or tuning parameters of HDFS that is, through the management console.

Refer Slide Time :(32:13)

HDFS Block Size

- Recall: impacts how much NameNode memory is used, number of map tasks that are showing up, and also have impacts on performance.
- Default 64 megabytes: Typically bumped up to 128 megabytes and can be changed based on workloads.
- The parameter that this changes dfs.blocksize or dfs.block.size.

Let us see, what are the, which are most important, which need to be decided for performance from, performance perspective. Now here, HDFS block size recall that, impacts how much name node memory is used, the number of map tasks that are showing up and also have the impacts on the performance. So, the by default the block size is 64 megabytes. And typically, it can go up to 128 megabytes and it can be changed based on the workloads. So, if let us say that, we want to have a better performance and the size, file size is too big, too large, then obviously more than 64 megabytes is good enough, so that so, so the parameter that this, make this particular changes is known as, DFS block size or DFS block dot size, where we have to mention about, the, the, the, the block size, by default it is 64 but we can increase up to, 128 megabytes. So if the, if the block size is more obviously, the number of blocks, will be, will be less and if it is less than, the amount of space which is required, to store in the namespace memory, will be less and also, if it is less and also the number of map tasks, which will be required to execute also, will be less. And so, basically there is a trade off, where the performance is required, so we have to set, this block size accordingly and application.

Refer Slide Time :(34:00)

HDFS Replication

- Default replication is 3.
- Parameter: `dfs.replication` ✓
- Tradeoffs:
 - Lower it to reduce replication cost
 - Less robust ✓
 - Higher replication can make data local to more workers
 - Lower replication → More space

So, another parameter is called the, 'HDFS', application by default that application is 3 and this parameter is set in a DFS replication, configuration file. And there is a trade-off, that means, if we lower, it to reduce the replication cost, that means, if the replication factor is not 3, if it is less, than the replication cost will be less. But, the trade-off is that, it will be less robust, robust in the sense, if some of the nodes are filled and there is only one replication, there is no replicas available of that node, so that particular data will not be available. So hence, it will be less robust. And also, the it will lose the performance, for example, if it is replicated then, it will be able to serve that, particular data block, from the closest possible, data block to the client. So, higher application can make data local to the more workers, lower replication means, and more space.

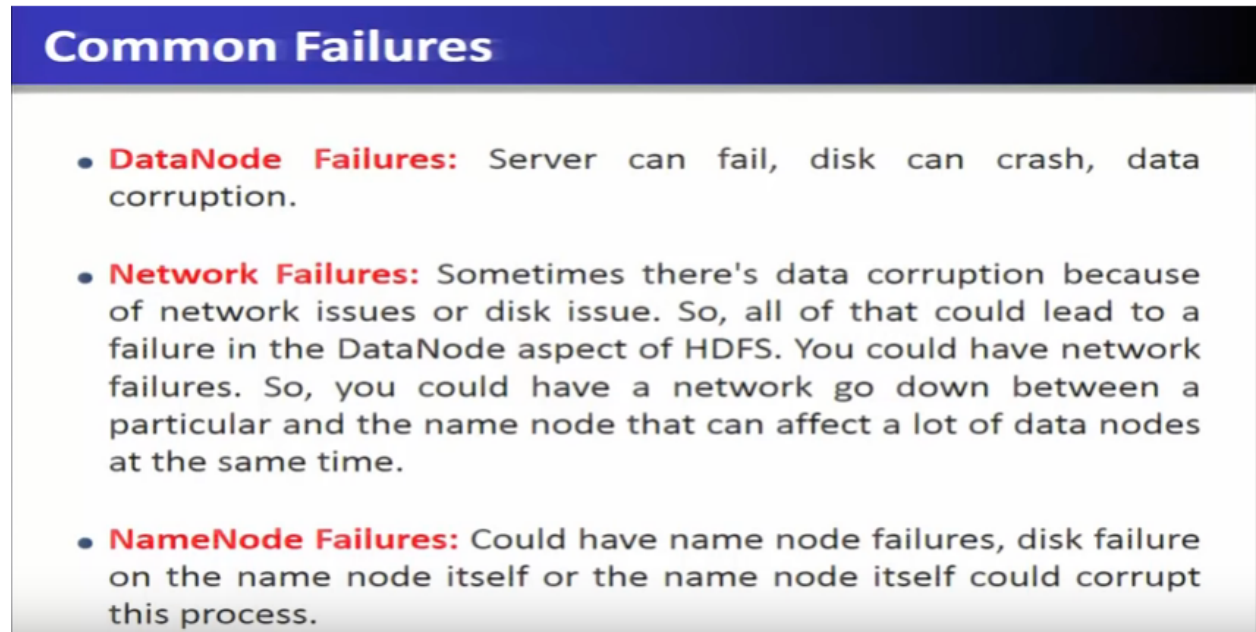
Refer Slide Time :(35:08)

Lot of other parameters

- Various tunables for datanode, namenode.
- **Examples:**
 - `Dfs.datanode.handler.count (10)`: Sets the number of server threads on each datanode
 - `Dfs.namenode.fs-limits.max-blocks-per-file`: Maximum number of blocks per file.
- **Full List:**
 - <http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml>

Lot of other parameters are there, which can be set. But, these two parameters which we have covered. And that is block size and application factor are the two most important, tunable parameters. So, the other such parameters are available, for example, DFS data node dot handler count is 10, that means, the number of the server threads, on each data nodes that is, maximum up to 10 and this is going to be a factor of this performance, of that data node operations. Similarly, there is another parameter which is called, 'Name Node'. Offense limits that is the maximum blocks per file, that is maximum number of blocks per file is also set, as per, this one.

Refer Slide Time :(36:01)



Common Failures

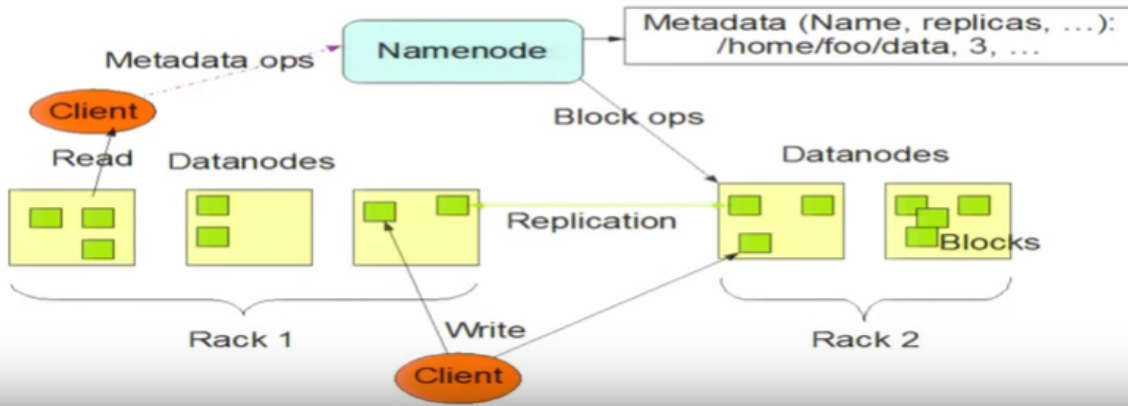
- **DataNode Failures:** Server can fail, disk can crash, data corruption.
- **Network Failures:** Sometimes there's data corruption because of network issues or disk issue. So, all of that could lead to a failure in the DataNode aspect of HDFS. You could have network failures. So, you could have a network go down between a particular and the name node that can affect a lot of data nodes at the same time.
- **NameNode Failures:** Could have name node failures, disk failure on the name node itself or the name node itself could corrupt this process.

So, let us see the, HDFS performance and its robustness. So, the common failures is a data node failure and the server can fail, disk and crash and the data also can become corrupt, in that case, the, the replicas is will be able to overcome, from this particular failures. another failure is called, 'Network Failure', sometimes there is a corruption of network or a disk issues. So, it could lead to the failure, of the data node in HDFS. So, when a network go down, then If, if let us say, it is replicas, replicas are across the, the rack then, it can be able to serve, from the other place. Similarly, name node if it is failed and it could be named node failure, disk failure, on the name node, on the name itself, it could corrupt this particular process. So, the Federation is there to, overcome from this name node failures.

Refer Slide Time :(37:09)

HDFS Robustness

- NameNode receives heartbeat and block reports from DataNodes



So, HDFS robustness, we have so far discussed. And so therefore the replication, on the data node is done. So, that it is a lag fault tolerant, that means, the replicas are across racks, so that if the one rack is down, it will be able to serve, from the other end. So, Name node receives the heartbeats and block the report from, this one data nodes, so all these is measured and wherever there is data note down, this information is captured or understood and the name node and that particular node is now, not being used by for the client, for the requests.

Refer Slide Time :(37:59)

Mitigation of common failures

- Periodic heartbeat: from DataNode to NameNode.
- **DataNodes without recent heartbeat:**
 - Mark the data. And any new I/O that comes up is not going to be sent to that data node. Also remember that NameNode has information on all the replication information for the files on the file system. So, if it knows that a datanode fails which blocks will follow that replication factor.
 - Now this replication factor is set for the entire system and also you could set it for particular file when you're writing the file. Either way, the NameNode knows which blocks fall below replication factor. And it will restart the process to re-replicate.

Handwritten notes:
 Name node
 Data node
 Data node

Now, the mitigation of common failures. So, periodic heartbeats from, data node to the name node is done that, we have seen and data nodes, without recent heartbeats is being marked. So mark the data and the new input, output, a new I/O that comes up is, not going to be sent to that node. Data node also remembers that, a name node has the information on all, the replication information, for the files, on the file system. So, if it knows that data, node fails which blocks, will follow that particular application factor. Now, this replication factor is set, for the entire system, so you could, set it for a particular file, when you are writing to a file either way, the name node knows, which block falls below the replication factor and it will restart the process to replicate, to read replicate. So, therefore let us see, this particular diagram that, this is the name node and it keeps on checking, the data nodes and several data nodes. So, these data nodes keeps on sending their, heartbeats at a periodic interval and by that, particular heartbeats, the name node knows that, these particular data nodes are active. If the heartbeats is not, received at the name node, name node, now, understand that this is down and if it is down then, this particular application factor is basically is reduced, for that, particular replicas stored on that data node. So, the name node knows which of that block falls, below the replication factor. And it will restart the process to re replicate. So, that number of, so that replication factor is maintained at all points of time. And that particular data, data node, which is down, which is detected as down, will not be used for, further usage. So, the migration of common failure is, handled by the name node, in this particular manner, with the help of the periodic heartbeats.

Refer Slide Time :(40:28)

Mitigation of common failures

- Checksum computed on file creation.
- Checksums stored in HDFS namespace.
- Used to check retrieved data.
- Re-read from alternate replica

Mitigation of other common failures, such as, checksum computed on a file, we shows that, the data or a block is corrupted or a checksum stored, in the HDFS namespace, also tells that it is failed. And used to check the retrieve data and reread the alternative, alternate replicas. So, that means that, whenever there is using checksum, if it is directed that, the data is replica is not or the data, which is accessed is, having an error using, some failure then, alternative replica is consulted up or is being accessed and then, it will be also made the, proper corrections, wherever there is a failure.

Refer Slide Time :(41:30)

Mitigation of common failures

- Multiple copies of central meta data structures.
- Failover to standby NameNode- manual by default.

So, multiple copies of Central Meta data structure is, being maintained to handle with these common failures. And failover, to standby the name node is there and normally it is manually done, by default.

Refer Slide Time :(41:47)

Performance

- Changing blocksize and replication factor can improve performance.
- **Example: Distributed copy**
- Hadoop distcp allows parallel transfer of files.

Now, let us see, the performance issue that, if we change the block size and the replica factor, replication factor, how is it going to improve the performance. Let us take an example, of a distributed copy operation, hadoop supports a distributed copy that, allows the parallel transfer of the files.

Refer Slide Time :(42:08)

Replication trade off with respect to robustness

- One performance tradeoff is, actually when you go out to do some of the map reduce jobs, having replicas gives additional locality possibilities, but the big trade off is the robustness. In this case, we said no replicas. Might lose a node or a local disk: can't recover because there is no replication. —
- Similarly, with data corruption, if you get a checksum that's bad, now you can't recover because you don't have a replica.
- Other parameters changes can have similar effects.

Now, there is a trade-off between the replication, trade-off with the respect to the, to the robustness. Before we start, the idea is that, if we reduce the replication factor, then it is going to affect to the robustness. For example, if let us say, it is not replicated, to the other data nodes. And if that data nodes, containing that data or a block fields, then it is not available at other end. Hence, it is going to affect the robustness. So, replication is so important, that we are going to discuss. So, one performance trade-off is actually, when you go out, to do some of the Map Reduce jobs, having replicas gives additional locality possibility. But, the big trade-off is the robustness, in this case we said, no replica, might lose a node or, a or a local discount recover because, there is no replica. Hence, if replication factor is, is not immutable, so if the replication so, no replica is available, if no replica is available, then obviously it is lead to a failure. And hence, there is no hence, it is not robust. Similarly, with the data corruption and if we get, the checks that is bad and we cannot recover, why because, we don't have any replicas and other parameters, changes have similar effects. So, basically there is a trade-off between, the replica and the robustness.

Refer Slide Time :(43:51)

Conclusion

- In this lecture, we have discussed design goals of HDFS, the read/write process to HDFS, the main configuration tuning parameters to control HDFS performance and robustness.

So in conclusion, in this lecture, we have discussed the HDFS. HDFS version two and operation that is read and write which is supported in HDFS, we have also seen, the main configuration, we have also seen the performance, parameters and the tuning parameters, with respect to the block size and the replication factor, to ensure about, the HDFS performance and its robustness trade off. Thank you.