BigData Planet

- Home
- Big Data
- Hadoop Tutorials
- Cassandra
- Hector API
- Request Tutorial
- About

LABELS: HADOOP-TUTORIAL, HDFS

3 OCTOBER 2013

Hadoop Tutorial: Part 1 - What is Hadoop ? (an Overview)

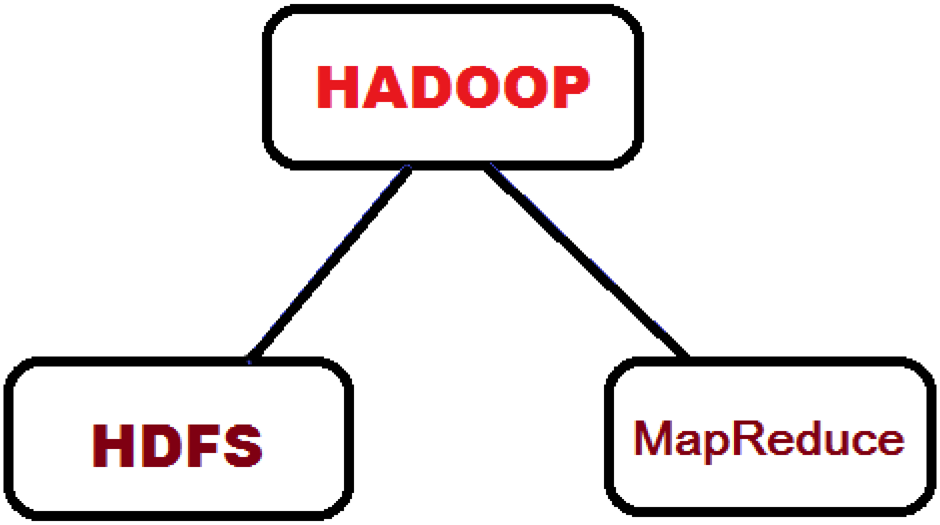
Hadoop is an open source software framework that supports data intensive distributed applications which is licensed under Apache v2 license.

At-least this is what you are going to find as the first line of definition on Hadoop in Wikipedia. So ***what is data intensive distributed applications?***

Well **data intensive** is nothing but **BigData** (data that has outgrown in size) and**distributed applications** are the applications that works on network by communicating and coordinating with each other by passing messages. (say using a RPC interprocess communication or through Message-Queue)

Hence Hadoop works on a distributed environment and is build to store, handle and process large amount of data set (in petabytes, exabyte and more). Now here since i am saying that hadoop stores petabytes of data, this doesn't mean that Hadoop is a database. Again remember its a framework that handles large amount of data for processing. You will get to know the difference between Hadoop and Databases (or NoSQL Databases, well that's what we call BigData's databases) as you go down the line in the coming tutorials.

Hadoop was derived from the research paper published by Google on *Google File System(GFS)* and *Google's MapReduce*. So there are two integral parts of Hadoop: **Hadoop Distributed File System(HDFS)** and **Hadoop MapReduce**.



Hadoop Distributed File System (HDFS)

HDFS is a filesystem designed for storing **very large files** with **streaming data access**patterns, running on clusters of **commodity hardware**.
Well Lets get into the details of the statement mentioned above:

Very Large files: Now when we say very large files we mean here that the size of the file will be in a range of gigabyte, terabyte, petabyte or may be more.

Streaming data access: HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, and then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

Commodity Hardware: Hadoop doesn't require expensive, highly reliable hardware. It's designed to run
on clusters of commodity hardware (commonly available hardware that can be obtained from multiple vendors) for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure.

Now here we are talking about a FileSystem, Hadoop Distributed FileSystem. And we all know about a few of the other File Systems like Linux FileSystem and Windows FileSystem. So the next question comes is...

What is the difference between normal FileSystem and Hadoop Distributed File System?

The major two differences that is notable between HDFS and other Filesystems are:

- **Block Size:** Every disk is made up of a block size. And this is the minimum amount of data that is written and read from a Disk.

Now a Filesystem also consists of blocks which is made out of these blocks on the disk. Normally disk blocks are of 512 bytes and those of filesystem are of a few kilobytes. In case of **HDFS** we also have the blocks concept. But here one block size is of 64 MB by default and which can be increased in an integral multiple of 64 i.e. 128MB, 256MB, 512MB or even more in GB's. It all depend on the requirement and use-cases.

So Why are these blocks size so large for HDFS? keep on reading and you will get it in a next few tutorials :)

- **Metadata Storage:** In normal file system there is a hierarchical storage of metadata i.e. lets say there is a folder *ABC***,** inside that folder there is again one another folder *DEF***,** and inside that there is *hello.txt* file. Now the information about *hello.txt* (i.e. metadata info of hello.txt) file will be with *DEF* and again the metadata of *DEF* will be with *ABC*. Hence this forms a hierarchy and this hierarchy is maintained until the root of the filesystem. But in **HDFS** we don't have a hierarchy of metadata. All the metadata information resides with a single machine known as ***Namenode*** (or Master Node) on the cluster. And this node contains all the information about other files and folder and lots of other information too, which we will learn in the next few tutorials. :)

Well this was just an overview of Hadoop and Hadoop Distributed File System. Now in the next part i will go into the depth of HDFS and there after MapReduce and will continue from here...

Let me know if you have any doubts in understanding anything into the comment section and i will be really glad to answer the same :)

*If you like what you just read and want to continue your learning on BIGDATA you can **subscribe to our***

Email** and Like our **facebook page

Back to Hadoop Tutorials

These might also help you :

1. Hadoop Tutorial: Part 4 - Write Operations in HDFS
2. Hadoop Tutorial: Part 3 - Replica Placement or Replication and Read Operations in HDFS
3. Hadoop Tutorial: Part 2 - Hadoop Distributed File System (HDFS)
4. Hadoop Tutorial: Part 1 - What is Hadoop ? (an Overview)
5. Best of Books and Resources to Get Started with Hadoop
6. Hadoop Tutorial: Part 5 - All Hadoop Shell Commands you will Need.
7. Hadoop Installation on Local Machine (Single node Cluster)

Find Comments below or Add one

Romain Rigaux said...

> Nice summary!

October 03, 2013

pragya khare said...

> I know i'm a beginner and this question myt be a silly 1....but can you please explain to me that how PARALLELISM is achieved via map-reduce at the processor level ??? if I've a dual core processor, is it that only 2 jobs will run at a time in parallel?

October 05, 2013

Anonymous said...

Hi I am from Mainframe background and with little knowledge of core java...Do you think Java is needed for learning Hadoop in addition to Hive/PIG ? Even want to learn Java for map reduce but couldn't find what all will be used in realtime..and definitive guide books seems tough for learning mapreduce with Java..any option where I can learn it step by step? Sorry for long comment..but it would be helpful if you can guide me..

October 05, 2013

Deepak Kumar said...

@Pragya Khare...
First thing always remember... the one Popular saying.... NO Questions are Foolish :) And btw it is a very good question.
Actually there are two things:

One is what will be the best practice? and other is what happens in there by default ?...

Well by default the number of mapper and reducer is set to 2 for any task tracker, hence one sees a maximum of 2 maps and 2 reduces at a given instance on a TaskTracker (which is configurable)..Well this Doesn't only depend on the Processor but on lots of other factor as well like ram, cpu, power, disk and others....

http://hortonworks.com/blog/best-practices-for-selecting-apache-hadoop-hardware/

And for the other factor i.e for Best Practices it depends on your use case. You can go through the 3rd point of the below link to understand it more conceptually

http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/

> Well i will explain all these when i will reach the advance MapReduce tutorials.. Till then keep reading !! :)

October 05, 2013

Deepak Kumar said...

> @Anonymous
> As Hadoop is written in Java, so most of its API's are written in core Java... Well to know about the Hadoop architecture you don't need Java... But to go to its API Level and start programming in MapReduce you need to know Core Java.
>
> And as for the requirement in java you have asked for... you just need simple core java concepts and programming for Hadoop and MapReduce..And Hive/PIG are the SQL kind of data flow languages that is really easy to learn...And since you are from a programming background it won't be very difficult to learn java :) you can also go through the link below for further details :)
>
> http://www.bigdataplanet.info/2013/09/What-are-the-Pre-requsites-for-getting-started-with-Big-Data-Technologies.html

October 05, 2013

Post a Comment

Newer Post→← Older Post

ABOUT THE AUTHOR

DEEPAK KUMAR

Big Data / Hadoop Developer, Software Engineer, Thinker, Learner, Geek, Blogger, Coder

I love to play around Data. Big Data !

g+

150 readers
BY FEEDBURNER

f take a look at my FB profile.

Subscribe updates via Email

Join BigData Planet to continue your learning on BigData Technologies

Subscribe

Get Updates on Facebook

Big Data Libraries

1. BIGDATA NEWS
2. CASSANDRA
3. HADOOP-TUTORIAL
4. HDFS
5. HECTOR-API
6. INSTALLATION
7. SQOOP

Which NoSQL Databases according to you is Most Popular ?

Get Connected on Google+

Most Popular Blog Article

- Hadoop Installation on Local Machine (Single node Cluster)
- Hadoop Tutorial: Part 5 - All Hadoop Shell Commands you will Need.

- What are the Pre-requisites for getting started with Big Data Technologies
- Hadoop Tutorial: Part 3 - Replica Placement or Replication and Read Operations in HDFS
- Hadoop Tutorial: Part 1 - What is Hadoop ? (an Overview)
- Hadoop Tutorial: Part 2 - Hadoop Distributed File System (HDFS)
- Hadoop Tutorial: Part 4 - Write Operations in HDFS
- Best of Books and Resources to Get Started with Hadoop
- How to use Cassandra CQL in your Java Application

Back to Top ▲

#Note: Use Screen Resolution of 1280 px and more to view the website @ its best. Also use the latest version of the browser as the website uses HTML5 and CSS3 :)

TwitterFacebookRSSGoogle

- ABOUT ME
- CONTACT
- PRIVACY POLICY

LABELS: HADOOP-TUTORIAL, HDFS

6 OCTOBER 2013

Hadoop Tutorial: Part 2 - Hadoop Distributed File System (HDFS)

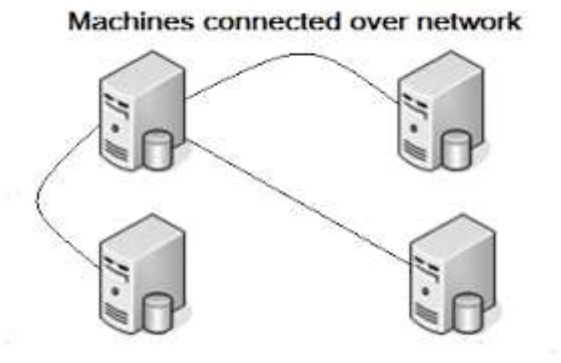
In the last tutorial on **What is Hadoop?** i have given you a brief idea about Hadoop. So the two integral parts of Hadoop is Hadoop **HDFS** and Hadoop **MapReduce**.

Lets go further deep inside HDFS.

Hadoop Distributed File System (HDFS) Concepts:

First take a look at the following two terminologies that will be used while describing HDFS.

Cluster: A hadoop cluster is made by having many machines in a network, each machine is termed as a node, and these nodes talks to each other over the network.



Block Size: This is the minimum amount of size of one block in a filesystem, in which data can be kept contiguously. *The default size of a single block in HDFS is 64 Mb.*

In HDFS, Data is kept by splitting it into small chunks or parts. Lets say you have a text file of 200 MB and you want to keep this file in a Hadoop Cluster. Then what happens is that, ***the file breaks or splits into a large number of chunks, where each chunk is equal to the block size that is set for the HDFS cluster (which is 64 MB by default).*** Hence a 200 Mb of file gets split into 4 parts, 3 parts of 64 mb and 1 part of 8 mb, and each part will be kept on a different machine. On which machine which split will be kept is decided by Namenode, about which we will be discussing in details below.

Now in a Hadoop Distributed File System or HDFS Cluster, there are two kinds of nodes, A Master Node and many Worker Nodes. These are known as:

Namenode (master node) and Datanode (worker node).

Namenode:

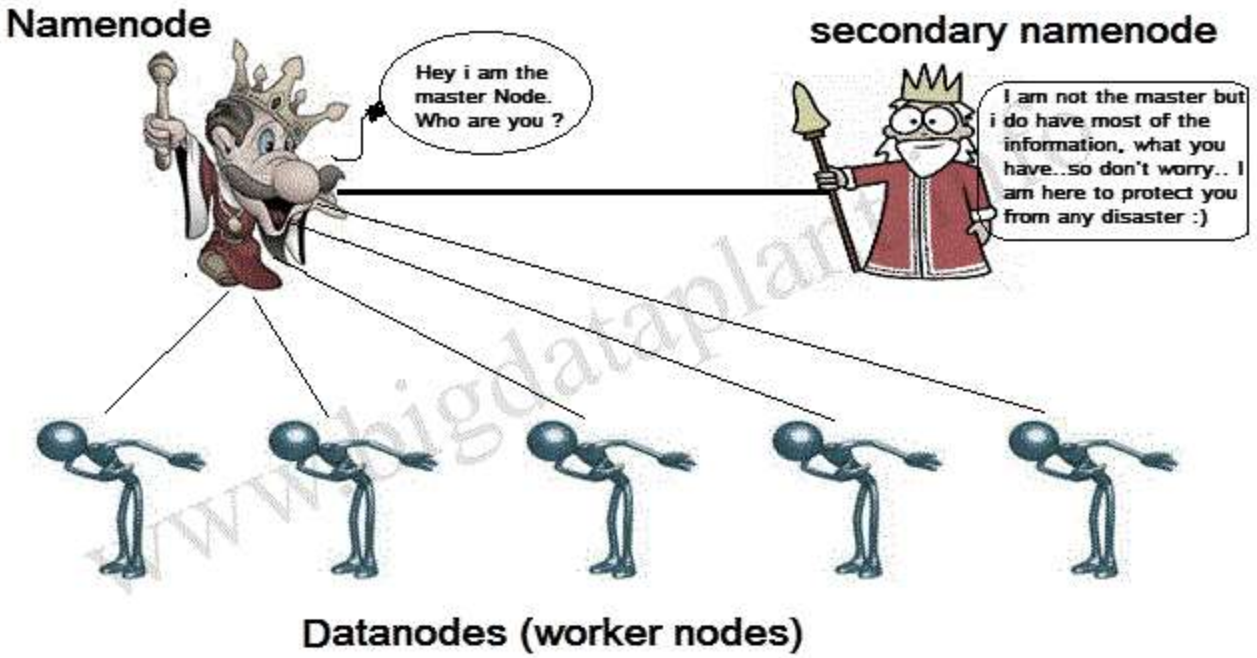
The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. So it contains the information of all the files, directories and their hierarchy in the cluster in the form of a **Namespace Image** and **edit logs**. Along with the filesystem information it also knows about the Datanode on which all the blocks of a file is kept.

A client accesses the filesystem on behalf of the user by communicating with the namenode and datanodes. The client presents a filesystem interface similar to a Portable Operating System Interface (POSIX), so the user code does not need to know about the namenode and datanode to function.

Datanode:

These are the workers that does the real work. And here by real work we mean that the storage of actual data is done by the data node. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.

Here one important thing that is there to note: *In one cluster there will be only one Namenode and there can be N number of datanodes.*


Namenode
Hey i am the master Node. Who are you ?
secondary namenode
I am not the master but i do have most of the information, what you have..so don't worry.. I am here to protect you from any disaster :)
Datanodes (worker nodes)

Since the Namenode contains the metadata of all the files and directories and also knows about the datanode on which each split of files are stored.

So lets say**Namenode goes down then what do you think will happen?**.

Yes, if the Namenode is Down we cannot access any of the files and directories in the cluster.
Even we will not be able to connect with any of the datanodes to get any of the files. Now think of it, since we have kept our files by splitting it in different chunks and also we have kept them in different datanodes. And it is the Namenode that keeps track of all the files metadata. So only Namenode knows how to reconstruct a file back into one from all the splits. and this is the reason that if Namenode is down in a hadoop cluster so every thing is down.
This is also the reason that's why Hadoop is known as a Single Point of failure.

Now since Namenode is so important, we have to make the namenode resilient to failure. And for that hadoop provides us with two mechanism.

The first way is to back up the files that make up the persistent state of the filesystem metadata. Hadoop can be configured so that the namenode writes its persistent state to multiple filesystems. These writes are synchronous and atomic. The usual configuration choice is to write to local disk as well as a remote NFS mount.

The second way is running a **Secondary Namenode.** Well as the name suggests, it **does not** act like a Namenode. So if it doesn't act like a namenode how does it prevents from the failure.

Well the **Secondary namenode** also contains a **namespace image** and **edit logs** like**namenode**. Now after every certain interval of time(which is one hour by default) it copies the **namespace image** from **namenode** and merge this **namespace image** with the **edit log** and copy it back to the **namenode** so that **namenode** will have the fresh copy of **namespace image**. Now lets suppose at any instance of time the **namenode**goes down and becomes corrupt then we can restart some other machine with the namespace image and the edit log that's what we have with the **secondary namenode**and hence can be prevented from a total failure.
Secondary Name node takes almost the same amount of memory and CPU

for its working as the Namenode. So it is also kept in a separate machine like that of a namenode. Hence we see here that ***in a single cluster we have one Namenode, one Secondary namenode and many Datanodes***, and HDFS consists of these three elements.

This was again an overview of Hadoop Distributed File System HDFS, In the next part of the tutorial we will know about the working of Namenode and Datanode in a more detailed manner.We will know how read and write happens in HDFS.

Let me know if you have any doubts in understanding anything into the comment section and i will be really glad to answer your questions :)

If you like what you just read and want to continue your learning on BIGDATA you can subscribe to our Email and Like our facebook page

Back to Hadoop Tutorials

These might also help you :

1. Hadoop Installation on Local Machine (Single node Cluster)
2. Hadoop Tutorial: Part 4 - Write Operations in HDFS
3. Hadoop Tutorial: Part 3 - Replica Placement or Replication and Read Operations in HDFS
4. Hadoop Tutorial: Part 2 - Hadoop Distributed File System (HDFS)
5. Hadoop Tutorial: Part 1 - What is Hadoop ? (an Overview)
6. Best of Books and Resources to Get Started with Hadoop

7. Hadoop Tutorial: Part 5 - All Hadoop Shell Commands you will Need.

Find Comments below or Add one

vishwash said...

very informative...

October 07, 2013

Tushar Karande said...

Thanks for such a informatic tutorials :)
please keep posting .. waiting for more... :)

October 08, 2013

Anonymous said...

Nice information........But I have one doubt like, what is the advantage of keeping the file in part of chunks on different-2 datanodes? What kind of benefit we are getting here?

October 08, 2013

Deepak Kumar said...

@Anonymous: Well there are lots of reasons... i will explain that with great details in the next few articles...
But for now let us understand this... since we have split the file into two, now we can take the power of two processors(parallel processing) on two different nodes to do our analysis(like search, calculation, prediction and lots more)..
Again lets say my file size is in some petabytes... Your won't find one Hard disk that big.. and lets say if it is there... how do you think that we are going to read and write on that hard disk(the latency will be really high to read and write)... it will take lots of time...Again there are more reasons for the same... I will make you understand this in more technical ways in the coming tutorials... Till then keep reading :)

October 08, 2013

Post a Comment

Newer Post→← Older Post

ABOUT THE AUTHOR

DEEPAK KUMAR

Big Data / Hadoop Developer, Software Engineer, Thinker, Learner, Geek, Blogger, Coder

I love to play around Data. Big Data !

150 readers
BY FEEDBURNER

take a look at my FB profile.

Subscribe updates via Email

Join BigData Planet to continue your learning on BigData Technologies

[ ] Subscribe

Get Updates on Facebook

Big Data Libraries

1. BIGDATA NEWS
2. CASSANDRA
3. HADOOP-TUTORIAL
4. HDFS

5. HECTOR-API

6. INSTALLATION

7. SQOOP

Which NoSQL Databases according to you is Most Popular ?

Get Connected on Google+

Most Popular Blog Article

- Hadoop Installation on Local Machine (Single node Cluster)
- Hadoop Tutorial: Part 5 - All Hadoop Shell Commands you will Need.
- What are the Pre-requisites for getting started with Big Data Technologies
- Hadoop Tutorial: Part 3 - Replica Placement or Replication and Read Operations in HDFS
- Hadoop Tutorial: Part 1 - What is Hadoop ? (an Overview)
- Hadoop Tutorial: Part 2 - Hadoop Distributed File System (HDFS)
- Hadoop Tutorial: Part 4 - Write Operations in HDFS
- Best of Books and Resources to Get Started with Hadoop
- How to use Cassandra CQL in your Java Application

Back to Top ▲

#Note: Use Screen Resolution of 1280 px and more to view the website @ its best. Also use the latest version of the browser as the website uses HTML5 and CSS3 :)

TwitterFacebookRSSGoogle

- ABOUT ME

- CONTACT

- PRIVACY POLICY

BigData Planet

- Home
- Big Data
- Hadoop Tutorials
- Cassandra
- Hector API
- Request Tutorial
- About

LABELS: HADOOP-TUTORIAL, HDFS

3 OCTOBER 2013

Hadoop Tutorial: Part 1 - What is Hadoop ? (an Overview)

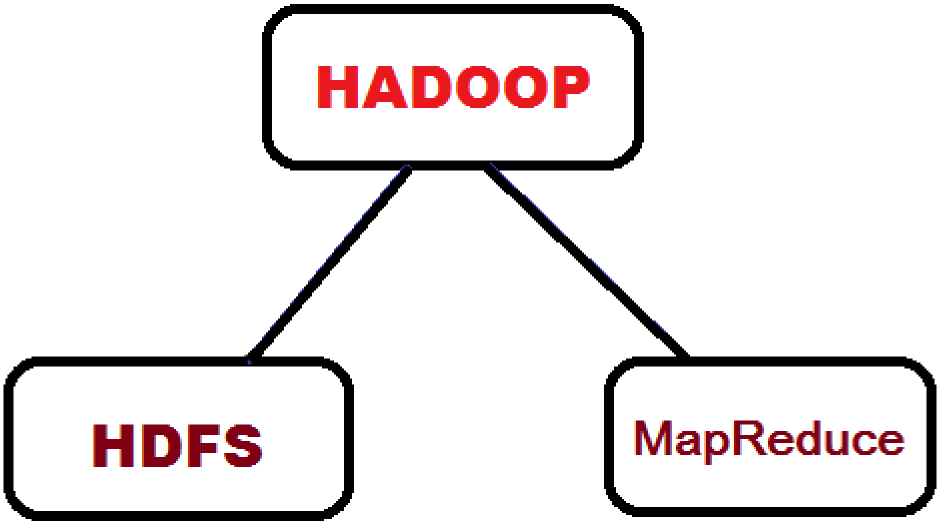
Hadoop is an open source software framework that supports data intensive distributed applications which is licensed under Apache v2 license.

At-least this is what you are going to find as the first line of definition on Hadoop in Wikipedia. So ***what is data intensive distributed applications?***

Well **data intensive** is nothing but **BigData** (data that has outgrown in size) and**distributed applications** are the applications that works on network by communicating and coordinating with each other by passing messages. (say using a RPC interprocess communication or through Message-Queue)

Hence Hadoop works on a distributed environment and is build to store, handle and process large amount of data set (in petabytes, exabyte and more). Now here since i am saying that hadoop stores petabytes of data, this doesn't mean that Hadoop is a database. Again remember its a framework that handles large amount of data for processing. You will get to know the difference between Hadoop and Databases (or NoSQL Databases, well that's what we call BigData's databases) as you go down the line in the coming tutorials.

Hadoop was derived from the research paper published by Google on *Google File System(GFS)* and *Google's MapReduce*. So there are two integral parts of Hadoop: **Hadoop Distributed File System(HDFS)** and **Hadoop MapReduce**.



Hadoop Distributed File System (HDFS)

HDFS is a filesystem designed for storing **very large files** with **streaming data access**patterns, running on clusters of **commodity hardware**.
Well Lets get into the details of the statement mentioned above:

Very Large files: Now when we say very large files we mean here that the size of the file will be in a range of gigabyte, terabyte, petabyte or may be more.

Streaming data access: HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, and then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

Commodity Hardware: Hadoop doesn't require expensive, highly reliable hardware. It's designed to run
on clusters of commodity hardware (commonly available hardware that can be obtained from multiple vendors) for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure.

Now here we are talking about a FileSystem, Hadoop Distributed FileSystem. And we all know about a few of the other File Systems like Linux FileSystem and Windows FileSystem. So the next question comes is...

What is the difference between normal FileSystem and Hadoop Distributed File System?

The major two differences that is notable between HDFS and other Filesystems are:

- **Block Size:** Every disk is made up of a block size. And this is the minimum amount of data that is written and read from a Disk. Now a Filesystem also consists of blocks which is made out of these blocks on the disk. Normally disk blocks are of 512 bytes and those of filesystem are of a few kilobytes. In case of **HDFS** we also have the blocks concept. But here one block size is of 64 MB by default and which can be increased in an integral multiple of 64 i.e. 128MB, 256MB, 512MB or even more in GB's. It all depend on the requirement and use-cases.

So Why are these blocks size so large for HDFS? keep on reading and you will get it in a next few tutorials :)

- **Metadata Storage:** In normal file system there is a hierarchical storage of metadata i.e. lets say there is a folder *ABC***,** inside that folder there is again one another folder *DEF***,** and inside that there is *hello.txt* file. Now the information about *hello.txt* (i.e. metadata info of hello.txt) file will be with *DEF* and again the metadata of *DEF* will be with *ABC*. Hence this forms a hierarchy and this hierarchy is maintained until the root of the filesystem. But in **HDFS** we don't have a hierarchy of metadata. All the metadata information resides with a single machine known as ***Namenode*** (or Master Node) on the cluster. And this node contains all the information about other files and folder and lots of other information too, which we will learn in the next few tutorials. :)

Well this was just an overview of Hadoop and Hadoop Distributed File System. Now in the next part i will go into the depth of HDFS and there after MapReduce and will continue from here...

Let me know if you have any doubts in understanding anything into the comment section and i will be really glad to answer the same :)

If you like what you just read and want to continue your learning on BIGDATA you can __subscribe to our Email__ and Like our __facebook page__

Back to Hadoop Tutorials

These might also help you :

1. Hadoop Tutorial: Part 4 - Write Operations in HDFS
2. Hadoop Tutorial: Part 3 - Replica Placement or Replication and Read Operations in HDFS
3. Hadoop Tutorial: Part 2 - Hadoop Distributed File System (HDFS)
4. Hadoop Tutorial: Part 1 - What is Hadoop ? (an Overview)
5. Best of Books and Resources to Get Started with Hadoop
6. Hadoop Tutorial: Part 5 - All Hadoop Shell Commands you will Need.
7. Hadoop Installation on Local Machine (Single node Cluster)

Find Comments below or Add one

Romain Rigaux said...

Nice summary!

October 03, 2013

pragya khare said...

I know i'm a beginner and this question myt be a silly 1....but can you please explain to me that how PARALLELISM is achieved via map-reduce at the processor level ??? if I've a dual core processor, is it that only 2 jobs will run at a time in parallel?

October 05, 2013

Anonymous said...

Hi I am from Mainframe background and with little knowledge of core java...Do you think Java is needed for learning Hadoop in addition to

> Hive/PIG ? Even want to learn Java for map reduce but couldn't find what all will be used in realtime..and definitive guide books seems tough for learning mapreduce with Java..any option where I can learn it step by step? Sorry for long comment..but it would be helpful if you can guide me..

October 05, 2013

Deepak Kumar said...

> @Pragya Khare...
> First thing always remember... the one Popular saying.... NO Questions are Foolish :) And btw it is a very good question.
> Actually there are two things:
>
> One is what will be the best practice? and other is what happens in there by default ?...
>
> Well by default the number of mapper and reducer is set to 2 for any task tracker, hence one sees a maximum of 2 maps and 2 reduces at a given instance on a TaskTracker (which is configurable)..Well this Doesn't only depend on the Processor but on lots of other factor as well like ram, cpu, power, disk and others....
>
> http://hortonworks.com/blog/best-practices-for-selecting-apache-hadoop-hardware/
>
> And for the other factor i.e for Best Practices it depends on your use case. You can go through the 3rd point of the below link to understand it more conceptually
>
> http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/
>
> Well i will explain all these when i will reach the advance MapReduce tutorials..
> Till then keep reading !! :)

October 05, 2013

Deepak Kumar said...

> @Anonymous
> As Hadoop is written in Java, so most of its API's are written in core Java... Well to know about the Hadoop architecture you don't need Java... But to go to its API Level and start programming in MapReduce you need to know Core Java.
>
> And as for the requirement in java you have asked for... you just need simple core java concepts and programming for Hadoop and MapReduce..And Hive/PIG are the SQL kind of data flow languages that is really easy to learn...And since you are from a programming background it won't be very difficult to learn java :) you can also go through the link below for further details :)
>
> http://www.bigdataplanet.info/2013/09/What-are-the-Pre-requsites-for-getting-started-with-Big-Data-Technologies.html

October 05, 2013

Post a Comment

Newer Post→← Older Post

ABOUT THE AUTHOR

DEEPAK KUMAR

Big Data / Hadoop Developer, Software Engineer, Thinker, Learner, Geek, Blogger, Coder

I love to play around Data. Big Data !

150 readers
BY FEEDBURNER

take a look at my FB profile.

Subscribe updates via Email

Join BigData Planet to continue your learning on BigData Technologies

[ ] Subscribe

Get Updates on Facebook

Big Data Libraries

1. BIGDATA NEWS
2. CASSANDRA
3. HADOOP-TUTORIAL
4. HDFS
5. HECTOR-API
6. INSTALLATION
7. SQOOP

Which NoSQL Databases according to you is Most Popular ?

Get Connected on Google+

Most Popular Blog Article

- Hadoop Installation on Local Machine (Single node Cluster)
- Hadoop Tutorial: Part 5 - All Hadoop Shell Commands you will Need.
- What are the Pre-requisites for getting started with Big Data Technologies
- Hadoop Tutorial: Part 3 - Replica Placement or Replication and Read Operations in HDFS

- Hadoop Tutorial: Part 1 - What is Hadoop ? (an Overview)
- Hadoop Tutorial: Part 2 - Hadoop Distributed File System (HDFS)
- Hadoop Tutorial: Part 4 - Write Operations in HDFS
- Best of Books and Resources to Get Started with Hadoop
- How to use Cassandra CQL in your Java Application

http://www.bigdataplanet.info/2013/10/Hadoop-Tutorial-Part-2-Hadoop-Distributed-File-System.html

http://www.devx.com/opensource/exploring-the-hadoop-distributed-file-system-hdfs.html

Specialized Dev Zones
eBook Library
.NET

[Java](#)
[C++](#)
[Web Dev](#)
[Architecture](#)
[Database](#)
[Security](#)
[Open Source](#)
[Enterprise](#)
[Mobile](#)
[Special Reports](#)
[10-Minute Solutions](#)
[DevXtra Blogs](#)

Related Resources

Create your own REST API Using OAuth Authentication
Microsoft Delivers New 64-Bit JIT Compiler for .NET
Bitcoin's True Purpose
Amazon Releases Its Own Chaos Gorilla
OpenTSDB Package Installation and Extractin
Time Series Data Points

Exploring the Hadoop Distributed File System (HDFS)

Kaushik Pal explores the basics of the Hadoop Distributed File System (HDFS), the underlying file system of the Apache Hadoop framework.

by Kaushik Pal

Nov 27, 2013

This article will explore the basics of the Hadoop Distributed File System (HDFS), the underlying file system of the Apache Hadoop framework. HDFS is a distributed storage space that spans across thousands of commodity hardware nodes. This file system provides fault tolerance, efficient throughput, streaming data access and reliability. The architecture of HDFS is suitable for storing a large volume of data and processing it quickly. HDFS is a part of Apache eco-system.

Introduction

Apache Hadoop is a software framework provided by the open source community. This is helpful in storing and processing of data-sets of large scale on clusters of commodity hardware. Hadoop is licensed under the Apache License 2.0.

The Apache Hadoop framework consists of the following modules:

- **Hadoop Common** – The common module contains libraries and utilities that are required by other modules of Hadoop.
- **Hadoop Distributed File System (HDFS)** – This is the distributed file-system that stores data on the commodity machines. This also provides a very high aggregate bandwidth across the cluster.
- **Hadoop YARN** – This is the resource management

platform that is responsible for managing compute resources over the clusters and using them for scheduling of users' applications.

- **Hadoop MapReduce** – This is the programming model used for large scale data processing.

All the modules in Hadoop are designed with a fundamental assumption that hardware failures (a single machine or entire rack) are obvious and thus should be automatically handled in the software application by the Hadoop framework. Apache Hadoop's HDFS components are originally derived from Google's MapReduce and Google File System (GFS) respectively.

Hadoop Distributed File System (HDFS)

HDFS is a primary distributed storage used by the Hadoop applications. An HDFS cluster primarily consists of a NameNode and the DataNode. The NameNode manages the file system metadata and DataNodes are used to store the actual data.

The HDFS architecture diagram explains the basic interactions among NameNode, the DataNodes, and the clients. The client's component calls the NameNode for file metadata or file modifications. The client then performs the actual file I/O operation directly with the DataNodes.

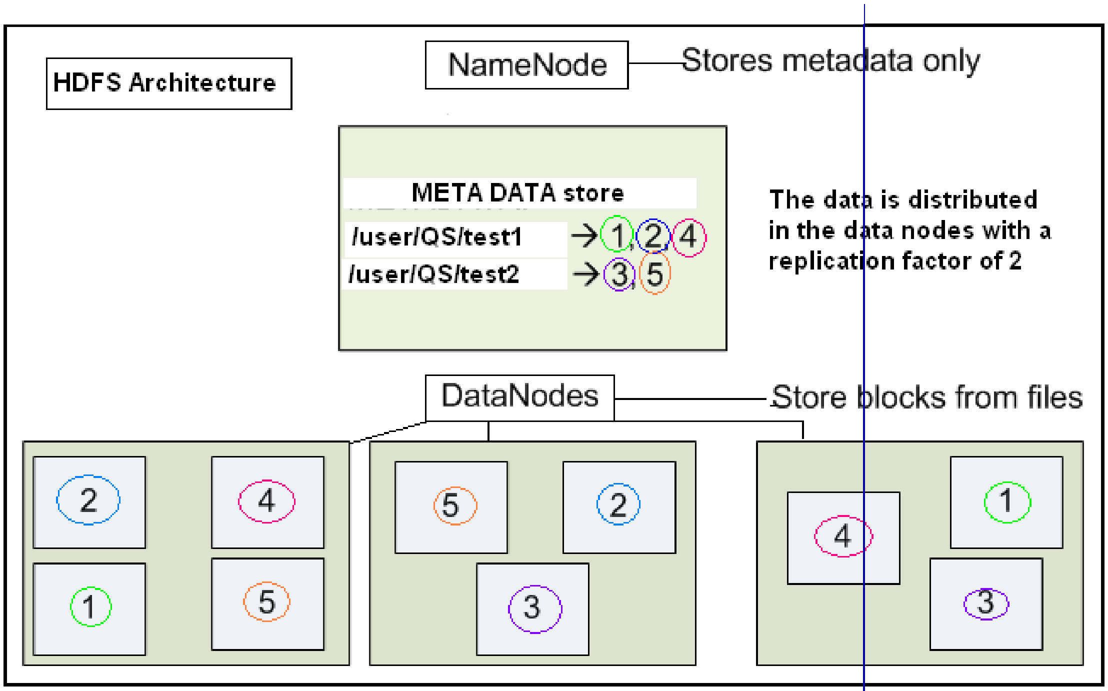


Figure 1: HDFS Architecture

Salient Features of HDFS

The following are some of the most important features:

- Hadoop, including HDFS, is a perfect match for distributed storage and distributed processing using low cost commodity hardware. Hadoop is scalable, fault tolerant and very simple to expand. MapReduce is well known for its simplicity and applicability in the case of large set of distributed applications.
- HDFS is highly configurable. The default configuration setup is good enough for most applications. In general, the default configuration needs to be tuned only for very large clusters.
- Hadoop is written based on the Java platform and is supported on nearly all major platforms.
- Hadoop supports shell and shell-like commands to communicate with HDFS directly.
- The NameNode and DataNodes have their own built in web servers that make it easy to check current status of the cluster.
- New features and updates are frequently implemented in HDFS. The following list is a subset

of the useful features available in HDFS:

- *File permissions* and authentication.
- *Rack awareness*: This helps to take a node's physical location into account while scheduling tasks and allocating storage.
- *Safemode*: This is the administrative tool mainly used for maintenance purposes.
- *fsck*: This is a utility used to diagnose the health of the file system and to find missing files or blocks.
- *fetchdt*: This is a utility used to fetch a DelegationToken and store it in a file on the local system.
- *Rebalancer*: This is a tool used to balance the cluster when the data is unevenly distributed across DataNodes.
- *Upgrade and rollback*: Once the software is upgraded, it is possible to roll back to the HDFS' state before the upgrade in case of any unexpected problem.
- *Secondary NameNode*: This node performs periodic checkpoints of the namespace and helps keep the size of file containing log of HDFS modifications within certain limits at the NameNode.
- *Checkpoint node*: This node performs periodic checkpoints of the namespace and helps minimize the size of the log stored at the NameNode containing changes made to the HDFS. It also replaces the role/function previously filled by the Secondary NameNode. As an alternative, the NameNode allows multiple nodes as check points, as long as there are no Backup nodes available (registered) with the system.
- *Backup node*: This can be defined as an extension to the Checkpoint node. Along with checkpointing, it is also used to receive a stream of edits from the NameNode. Thus it maintains its own in-memory copy of the namespace. It is always in sync with the active NameNode and namespace state. Only one Backup node is allowed to be registered with the NameNode at a time.

Goal of HDFS

Hadoop has a goal to use commonly available servers in a very large cluster, where each and every server has a set of inexpensive internal disk drives. For better performance, the MapReduce API tries to assign the workloads on these servers where the data is stored to be processed. This is known as *data locality*. Because of this, in a Hadoop environment, it is not recommended to use a storage area network (SAN), or a network attached storage (NAS). For Hadoop deployments using a SAN or NAS, the extra network communica¬tion overhead can cause performance bottlenecks, especially in case of larger clus¬ters.

Now, consider a situation in which we have a cluster of 1000-machines, and each of these machines has three internal disk drives. Think of the failure rate of a cluster composed of 3000 inexpensive drives + 1000 inexpensive servers! The component mean time to failure (MTTF) you are going to experience in a Hadoop cluster is likely similar to the zipper on your kid's jacket - it is bound to fail. The best part about Hadoop is that the reality of the MTTF rates associated with inexpen¬sive hardware is actually well understood and accepted.

This forms a part of the strength of Hadoop. Hadoop has built-in fault tolerance and fault-compensation capabilities. The same goes for HDFS, as the data is divided into blocks and chunks, and copies of these chunks/blocks are stored on other servers across the Ha¬doop cluster.

Case Study

Let us consider a file that contains the telephone numbers of all the residents in the United States of America. Those who have their last starting name with *A* could be stored on *server 1*; people having their last name begin with *B* are on *server 2*, and so on.

In a Hadoop environment, pieces of this phonebook would be stored and distributed on the entire cluster. To reconstruct the data of the entire phonebook, your program would need access the blocks from every server in the cluster. To achieve higher availability, HDFS replicates smaller pieces of data onto two additional servers by default. One can talk about redundancy here but the argument to support redundancy is to avoid the failure condition and provide a fault tolerance solution.

This redundancy can be increased or decreased on a per-file basis or for the whole environment. This redundancy offers multiple benefits. The most obvious being that the data is highly available. In addition to this, the data redundancy allows the Hadoop cluster to break work up into smaller chunks and run those smaller jobs on all the servers in the cluster for better scalability. Finally, as an end user we get the benefit of data locality, which is critical while working with large data sets.

Conclusion

We have seen that HDFS is one of the major components in the Apache Hadoop eco-system. The file system is the underlying storage structure, which is very powerful compared to the local file system.

Hope you have enjoyed the article and understood the basic concepts of HDFS. Keep reading.

About the Author

Kaushik Pal *is a technical architect with 15 years of experience in enterprise application and product development. He has expertise in web technologies, architecture/design, java/j2ee, Open source and big data technologies. You can find more of his work atwww.techalpine.com and you can email him here.*

E-mail the Author

Other Articles by This Author

0 Comments (click to add your comment)

Comment and Contribute

Your name/nickname

Your email

WebSite

Subject

(Maximum characters: 1200). You have 1200 characters left.

Privacy & Terms

Submit Your Comment

Sitemap

http://beginnersbook.com/2013/05/hdfs/

- HOME
- CONTACT US
- CORE JAVA
- JSP
- JSTL
- SQL
- JAVA COLLECTIONS
- SEO
- WORDPRESS
- C
- INTERVIEW Q&A
- OOPS CONCEPTS

- SERVLET

Hadoop Distributed File System(HDFS)

by CHAITANYA SINGH

in HADOOP

Before understanding what is **HDFS** first I would like to explain what is distributed file system.

What is Distributed File System?

As you know that each physical system has its own storage limit. And when it comes to store lots of data then we may need more than one system, Basically a network of systems. So that the data can be segregated among various machines which are connected to each other through a network. Such type of management in order to store bulk of data is known as **distributed file system**.

What is HDFS – Hadoop Distributed File System?

Hadoop has its own distributed file system which is known as HDFS (renamed from NDFS).

HDFS Design

1. Hadoop doesn't requires expensive hardware to store data, rather it is designed to support common and easily available hardware.
2. It is designed to store very very large file(As you all know that in order to index whole web it may require to store files which are in terabytes and petabytes or even more than that). **Hadoop clusters** are used to perform this task.
3. It is designed for streaming data access.

Hadoop file system

1) **Local:** This file system is for locally connected disks.

2) **HDFS:** Hadoop distributed file system: Explained above

3) **HFTP:** The purpose of it to provide read-only access for **Hadoop distributed file system** over HTTP.

4) **HSFTP:** It is almost similar to HFTP, unlike HFTP it provides read-only on**HTTPS.**

5) **HAR – Hadoop's Archives:** Used for archiving files.

6) **WebHDFS:** Grant write access on HTTP.

7) **KFS:** Its a cloud store system similar to GFS and HDFS.

8) **Distributed RAID:** Like HAR it is also used for archival.

9) **S3:** A file system provided by Amazon S3

HDFS Cluster Nodes

HDFS cluster has two nodes:

1. namenode
2. datanode

1) Namenodes

It basically stores the name and addresses of **datanodes**. It stores the data in form of a tree. Without Namenodes this whole system of stroing and retrieving data would not work as it is responsible to know which data is stored where.

2) Datanodes

Datanodes are used to store the data in form of **blocks.** They store and retrieve data in form of data blocks after communication with Namenodes.

Important links:

1. HDFS Guide
2. HDFS Java APi
3. HDFS Source code

You might like:

1. Hadoop tutorial
2. File I/O in C programming with examples
3. How to edit .htaccess file in WordPress
4. How to prevent access to .htaccess – Make it more secure
5. How to create a File in Java
6. How to write to a file in java using FileOutputStream
7. Append to a file in java using BufferedWriter, PrintWriter, FileWriter

{ 0 comments… add one now }

Leave a Comment

Name *

E-mail *

Website

□ Notify me of followup comments via e-mail

Submit

□ Confirm you are NOT a spammer

- POPULAR TUTORIALS
  - Core Java Tutorial
  - JSP Tutorial
  - JSTL Tutorial
  - Java Collections Tutorial
  - Servlet Tutorial
  - C Tutorial
-

SEARCH THIS SITE

-

To search, type and I

- FOLLOW ME ON GOOGLE+
-
-
-
- CONNECT WITH US ON FACEBOOK
-
- JOIN US ON GOOGLE PLUS
-

Apache > Hadoop > Core > docs > r1.0.4

Search the site with googl Search

- Project

- Wiki
- **Hadoop 1.0.4 Documentation**

Last Published: 02/13/2013 19:20:58

Getting Started
Guides
MapReduce
HDFS
HDFS Users
HDFS Architecture
Permissions
Quotas
Synthetic Load Generator
WebHDFS REST API
C API libhdfs
Common
Miscellaneous

PDF

HDFS Architecture Guide

Introduction

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject. The project URL ishttp://hadoop.apache.org/hdfs/.

Assumptions and Goals

Hardware Failure

Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

Streaming Data Access

Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. POSIX imposes many hard requirements that are not needed for applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

Large Data Sets

Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

Simple Coherency Model

HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

“Moving Computation is Cheaper than Moving Data”

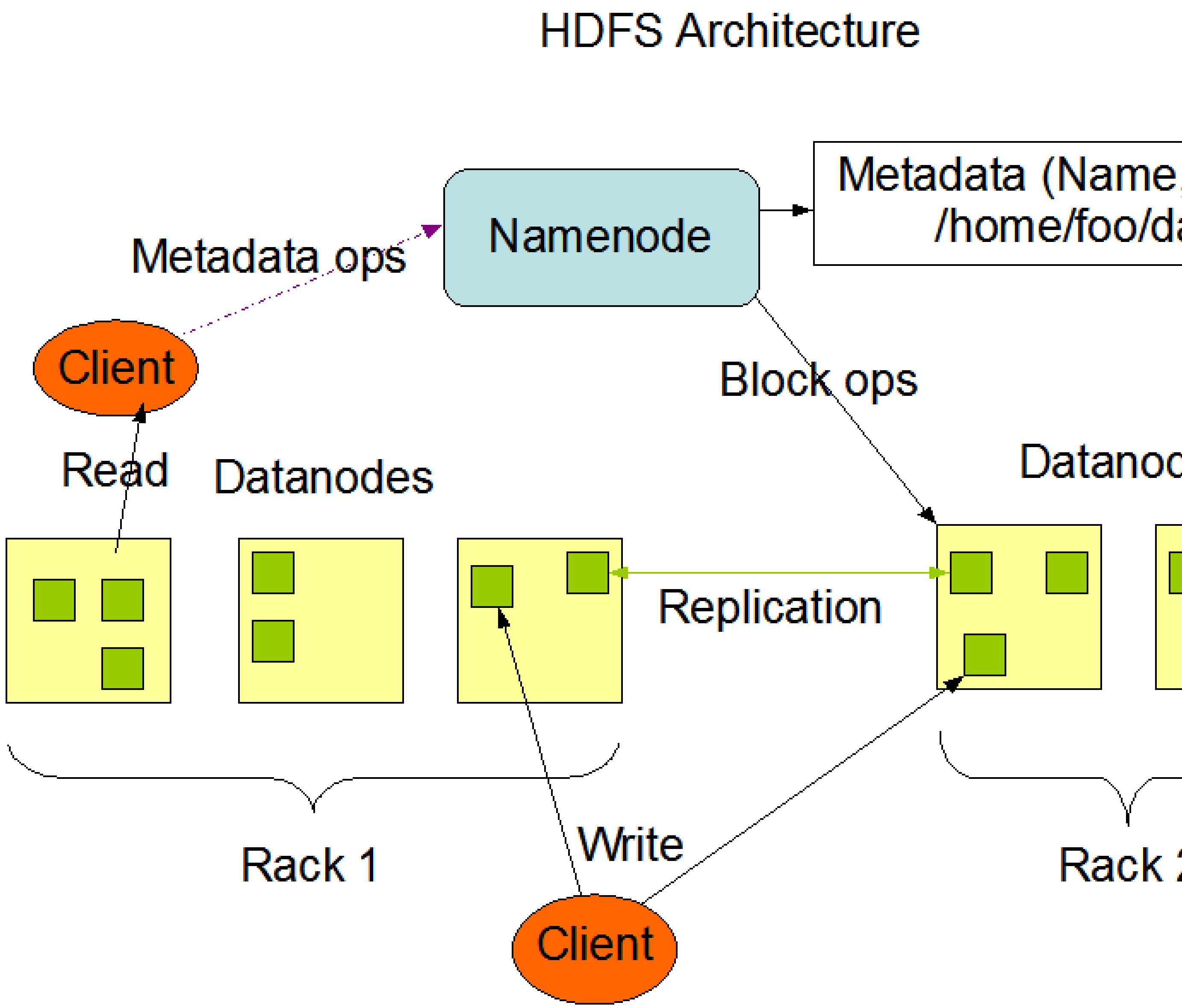
A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

Portability Across Heterogeneous Hardware and Software Platforms

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system’s clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.



The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

The File System Namespace

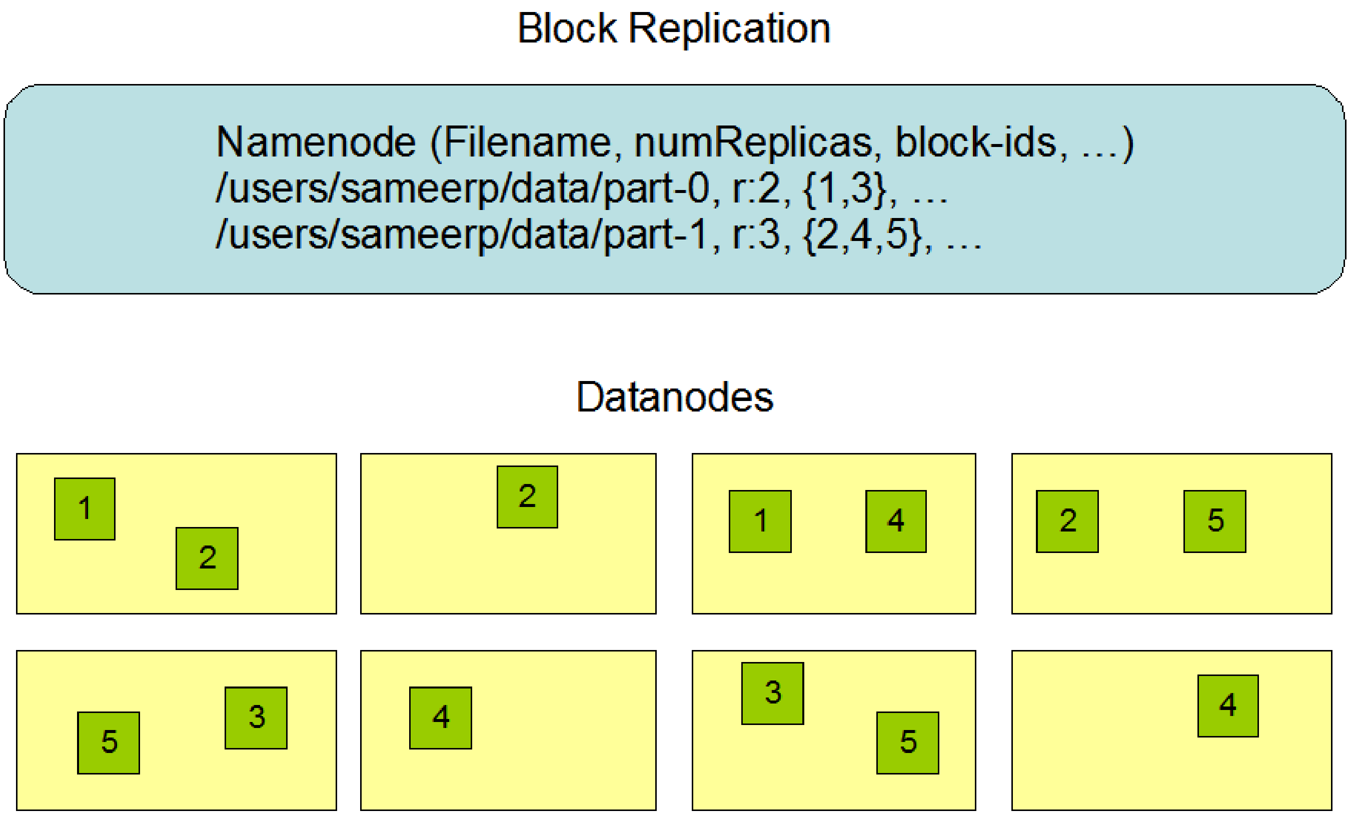
HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. HDFS does not yet implement user quotas. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.

The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.


Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Replica Placement: The First Baby Steps

The placement of replicas is critical to HDFS reliability and performance. Optimizing replica placement distinguishes HDFS from most other distributed file systems. This is a feature that needs lots of tuning and experience. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization. The current implementation for the replica placement policy is a first effort in this direction. The short-term goals of implementing this policy are to validate it on production systems, learn more about its behavior, and build a foundation to test and research more sophisticated policies.

Large HDFS instances run on a cluster of computers that commonly spread across many racks. Communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks.

The NameNode determines the rack id each DataNode belongs to via the process outlined in Hadoop Rack Awareness. A simple but non-optimal policy is to place replicas on unique racks. This prevents losing data when an entire rack fails and allows use of bandwidth from multiple racks when reading data. This policy evenly distributes replicas in the cluster which makes it easy to balance load on component failure. However, this policy increases the cost of writes because a write needs to transfer blocks to multiple racks.

For the common case, when the replication factor is three, HDFS’s placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure; this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. One third of replicas are on one node, two thirds of replicas are on one rack, and the other third are evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance.

The current, default replica placement policy described here is a work in progress.

Replica Selection

To minimize global bandwidth consumption and read latency, HDFS tries to satisfy a read request from a replica that is closest to the reader. If there exists a replica on the same rack as the reader node, then that replica is preferred to satisfy the read request. If angg/ HDFS cluster spans multiple data centers, then a replica that is resident in the local data center is preferred over any remote replica.

Safemode

On startup, the NameNode enters a special state called Safemode. Replication of data blocks does not occur when the NameNode is in the Safemode state. The NameNode receives Heartbeat and Blockreport messages from the DataNodes. A Blockreport contains the list of data blocks that a DataNode is hosting. Each block has a specified minimum number of replicas. A block is considered safely replicated when the minimum number of replicas of that data block has checked in with the NameNode. After a configurable percentage of safely replicated data blocks checks in with the NameNode (plus an additional 30 seconds), the NameNode exits the Safemode state. It then determines the list of data blocks (if any) that still have fewer than the specified number of replicas. The NameNode then replicates these blocks to other DataNodes.

The Persistence of File System Metadata

The HDFS namespace is stored by the NameNode. The NameNode uses a transaction log called the EditLog to persistently record every change that occurs to file system metadata. For example, creating a new file in HDFS causes the NameNode to insert a record into the EditLog indicating this. Similarly, changing the replication factor of a file causes a new record to be inserted into the EditLog. The NameNode uses a file in its local host OS file system to store the EditLog. The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in a file called the FsImage. The FsImage is stored as a file in the NameNode’s local file system too.

The NameNode keeps an image of the entire file system namespace and file Blockmap in memory. This key metadata item is designed to be compact, such that a NameNode with 4 GB of RAM is plenty to support a huge number of files and directories. When the NameNode starts up, it reads the FsImage and EditLog from disk, applies all the transactions from the EditLog to the in-memory representation of the FsImage, and flushes out this new version into a new FsImage on disk. It can then truncate the old EditLog because its transactions have been applied to the persistent FsImage. This process is called a checkpoint. In the current implementation, a checkpoint only occurs when the NameNode starts up. Work is in progress to support periodic checkpointing in the near future.

The DataNode stores HDFS data in files in its local file system. The DataNode has no knowledge about HDFS files. It stores each block of HDFS data in a separate file in its local file system. The DataNode does not create all files in the same directory. Instead, it uses a heuristic to determine the optimal number of files per directory and creates subdirectories appropriately. It is not optimal to create all local files in the same directory because the local file system might not be able to efficiently support a huge number of files in a single directory. When a DataNode starts up, it scans through its local file system, generates a list of all HDFS data blocks that correspond to each of these local files and sends this report to the NameNode: this is the Blockreport.

The Communication Protocols

All HDFS communication protocols are layered on top of the TCP/IP protocol. A client establishes a connection to a configurable TCP port on the NameNode machine. It talks the ClientProtocol with the NameNode. The DataNodes talk to the NameNode using the DataNode Protocol. A Remote Procedure Call (RPC) abstraction wraps both the Client Protocol and the DataNode Protocol. By design, the NameNode never initiates any RPCs. Instead, it only responds to RPC requests issued by DataNodes or clients.

Robustness

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any newIO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

Cluster Rebalancing

The HDFS architecture is compatible with data rebalancing schemes. A scheme might automatically move data from one DataNode to another if the free space on a DataNode falls below a certain threshold. In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster. These types of data rebalancing schemes are not yet implemented.

Data Integrity

It is possible that a block of data fetched from a DataNode arrives corrupted. This corruption can occur because of faults in a storage device, network faults, or buggy software. The HDFS client software implements checksum checking on the contents of HDFS files. When a client creates an HDFS file, it computes a checksum of each block of the file and stores these checksums in a separate hidden file in the same HDFS namespace. When a client retrieves file contents it verifies that the data it received from each DataNode matches the checksum stored in the associated checksum file. If not, then the client can opt to retrieve that block from another DataNode that has a replica of that block.

Metadata Disk Failure

The FsImage and the EditLog are central data structures of HDFS. A corruption of these files can cause the HDFS instance to be non-functional. For this reason, the NameNode can be configured to support maintaining multiple copies of the FsImage and EditLog. Any update to either the FsImage or EditLog causes each of the FsImages and EditLogs to get updated synchronously. This synchronous updating of multiple copies of the FsImage and EditLog may degrade the rate of namespace transactions per second that a NameNode can support. However, this degradation is acceptable because even though HDFS applications are very data intensive in nature, they are not metadata intensive. When a NameNode restarts, it selects the latest consistent FsImage and EditLog to use.

The NameNode machine is a single point of failure for an HDFS cluster. If the NameNode machine fails, manual intervention is necessary. Currently, automatic restart and failover of the NameNode software to another machine is not supported.

Snapshots

Snapshots support storing a copy of data at a particular instant of time. One usage of the snapshot feature may be to roll back a corrupted HDFS instance to a previously known good point in time. HDFS does not currently support snapshots but will in a future release.

Data Organization

Data Blocks

HDFS is designed to support very large files. Applications that are compatible with HDFS are those that deal with large data sets. These applications write their data only once but they read it one or more times and require these reads to be satisfied at streaming speeds. HDFS supports write-once-read-many semantics on files. A typical block size used by HDFS is 64 MB. Thus, an HDFS file is chopped up into 64 MB chunks, and if possible, each chunk will reside on a different DataNode.

Staging

A client request to create a file does not reach the NameNode immediately. In fact, initially the HDFS client caches the file data into a temporary local file. Application writes are transparently redirected to this temporary local file. When the local file accumulates data worth over one HDFS block size, the client contacts the NameNode. The NameNode inserts the file name into the file system hierarchy and allocates a data block for it. The NameNode responds to the client request with the identity of the DataNode and the destination data block. Then the client flushes the block of data from the local temporary file to the specified DataNode. When a file is closed, the remaining un-flushed data in the temporary local file is transferred to the DataNode. The client then tells the NameNode that the file is closed. At this point, the NameNode commits the file creation operation into a persistent store. If the NameNode dies before the file is closed, the file is lost.

The above approach has been adopted after careful consideration of target applications that run on HDFS. These applications need streaming writes to files. If a client writes to a remote file directly without any client side buffering, the network speed and the congestion in the network impacts throughput considerably. This approach is not without precedent. Earlier distributed file systems, e.g. AFS, have used client side caching to improve performance. A POSIX requirement has been relaxed to achieve higher performance of data uploads.

Replication Pipelining

When a client is writing data to an HDFS file, its data is first written to a local file as explained in the previous section. Suppose the HDFS file has a replication factor of three. When the local file accumulates a full block of user data, the client retrieves a list of DataNodes from the NameNode. This list contains the DataNodes that will host a replica of that block. The client then flushes the data block to the first DataNode. The first DataNode starts receiving the data in small portions (4 KB), writes each portion to its local repository and transfers that portion to the second DataNode in the list. The second DataNode, in turn starts receiving each portion of the data block, writes that portion to its repository and then flushes that portion to the third DataNode. Finally, the third DataNode writes the data to its local repository. Thus, a DataNode can be receiving data from the previous one in the pipeline and at the same time forwarding data to the next one in the pipeline. Thus, the data is pipelined from one DataNode to the next.

Accessibility

HDFS can be accessed from applications in many different ways. Natively, HDFS provides a Java API for applications to use. A C language wrapper for this Java API is also available. In addition, an HTTP browser can also be used to browse the files of an HDFS instance. Work is in progress to expose HDFS through the WebDAV protocol.

FS Shell

HDFS allows user data to be organized in the form of files and directories. It provides a commandline interface called FS shell that lets a user interact with the data in HDFS. The syntax of this command set is similar to other shells (e.g. bash, csh) that users are already familiar with. Here are some sample action/command pairs:

Action	Command
Create a directory named `/foodir`	`bin/hadoop dfs -mkdir /foodir`
Remove a directory named `/foodir`	`bin/hadoop dfs -rmr /foodir`
View the contents of a file named `/foodir/myfile.txt`	`bin/hadoop dfs -cat /foodir/myfile`

FS shell is targeted for applications that need a scripting language to interact with the stored data.

DFSAdmin

The DFSAdmin command set is used for administering an HDFS cluster. These are commands that are used only by an HDFS administrator. Here are some sample action/command pairs:

Action	Command
Put the cluster in Safemode	`bin/hadoop dfsadmin -safemode enter`
Generate a list of DataNodes	`bin/hadoop dfsadmin -report`
Recommission or decommission DataNode(s)	`bin/hadoop dfsadmin -refreshNodes`

Browser Interface

A typical HDFS install configures a web server to expose the HDFS namespace through a configurable TCP port. This allows a user to navigate the HDFS namespace and view the contents of its files using a web browser.

Space Reclamation

File Deletes and Undeletes

When a file is deleted by a user or an application, it is not immediately removed from HDFS. Instead, HDFS first renames it to a file in the `/trash` directory. The file can be restored quickly as long as it remains in `/trash`. A file remains in`/trash` for a configurable amount of time. After the expiry of its life in `/trash`, the NameNode deletes the file from the HDFS namespace. The deletion of a file causes the blocks associated with the file to be freed. Note that there could be an appreciable time delay between the time a file is deleted by a user and the time of the corresponding increase in free space in HDFS.

A user can Undelete a file after deleting it as long as it remains in the `/trash` directory. If a user wants to undelete a file that he/she has deleted, he/she can navigate the `/trash` directory and retrieve the file. The `/trash` directory contains only the latest copy of the file that was deleted. The `/trash` directory is just like any other directory with one special feature: HDFS applies specified policies to automatically delete files from this directory. The current default policy is to delete files from `/trash` that are more than 6 hours old. In the future, this policy will be configurable through a well defined interface.

Decrease Replication Factor

When the replication factor of a file is reduced, the NameNode selects excess replicas that can be deleted. The next Heartbeat transfers this information to the DataNode. The 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.

References

HDFS Java API: http://hadoop.apache.org/core/docs/current/api/

HDFS source code: http://hadoop.apache.org/hdfs/version_control.html

by Dhruba Borthakur

Last Published: 02/13/2013 19:20:58

- LearnNowOnline Home

Official Blog

TAG ARCHIVES: HADOOP DISTRIBUTED FILE SYSTEM

The Power of Hadoop

Leave a reply

Even within the context of other hi-tech technologies, Hadoop went from obscurity to fame in a miraculously short about of time. It had to… the pressures driving the development of this technology were too great. If you are not familiar with Hadoop, let's start by looking at the void it is trying to fill.

Companies, up until recently—say the last five to ten years or so—did not have the massive amounts of data to manage as they do today. Most companies only had to manage the data relating to running their business and managing their customers. Even those with millions of customers didn't have trouble storing data using your everyday relational database like Microsoft SQL Server or Oracle.

But today, companies are realizing that with the growth of the Internet and with self-servicing (or SaaS) Web sites, there are now hundreds of millions of potential customers that are all voluntarily providing massive amounts of valuable business intelligence. Think of storing something as simple as a Web log that provides every click of every user on your site. How does a company store and manipulate this data when it is generating potentially trillions of rows of data every year?

Generally speaking, the essence of the problem Hadoop is attempting to solve is that data is coming in faster than hard drive capacities are growing. Today we have 4 TB drives available which can then be assembled on SAN or NAS devices to easily get 40 TB volumes or maybe even 400 TB volumes. But what if you needed a 4,000 TB or 4 Petabytes (PB) volume? The costs quickly get incredibly high for most companies to absorb…until now. Enter Hadoop.

Hadoop Architecture

One of the keys to Hadoop's success is that it operates on everyday common hardware. A typical company has a backroom with hardware that has since past its prime. Using old and outdated computers, one can pack them full of relatively inexpensive hard drives (doesn't need to be the same total capacity within each computer) and use them within a Hadoop cluster. Need to expand capacity? Add more computers or hard drives. Hadoop can leverage all the hard drives into one giant volume available for storing all types of data, from web logs to large video files. It is not uncommon for Hadoop to be used to store rows of data that are over 1GB per row!

The file system that Hadoop uses is called the Hadoop Distributed File System or HDFS. It is a highly fault tolerant file system that focuses on high availability and fast readabilities. It is best used for data that is written once and read often. It leverages all the hard drives in the systems when writing data because Hadoop knows that bottlenecks stem from writing and reading to a single hard drive. The more hard drives are used simultaneously during the writing and reading of data, the faster the system operates as a whole.

The HDFS file system operates in small file blocks which are spread across all hard drives available within a cluster. The block size is configurable and optimized to the data being stored. It also replicates the blocks over multiple drives across multiple computers and even across multiple network subnets. This allows for hard drives or computers to fail (and they will) and not disrupt the system. It also allows Hadoop to be strategic in which blocks it accesses during a read. Hadoop will choose to read certain replicated blocks when it feels it can retrieve the data faster using one computer over another. Hadoop analyses which computers and hard drives are currently being utilized, along with network bandwidth, to strategically pick the next hard drive to read a block. This produces a system that is very quick to respond to requests.

MapReduce

Despite the relatively odd name, MapReduce is the cornerstone of Hadoop's data retrieval system. It is an abstracted programming layer on top of HDFS and is responsible for simplifying how data is read back to the user. It has a purpose similar to SQL in that it allows programmers to focus on building intelligent queries and not get involved in the underlying plumbing responsible for implementing or optimizing the queries. The “Map” part of the name refers to the task of building a map on the best way to sort and filter the information requested and then to return it as a pseudo result set. The “Reduce” task summarizes the data like the counting and summing of certain columns.

These two tasks are both analyzed by the Hadoop engine and then broken into many pieces or nodes (a divide and conquer model) which are all processed in parallel by individual workers. This result is the ability to process Petabytes of data in a matter of hours.

MapReduce is an open source project originally developed by Google and has been now ported over to many programming languages. You can find out more on MapReduce by visitinghttp://mapreduce.org.

In my next post, I’ll take a look at some of the other popular components around Hadoop, including advanced analytical tools like Hive and Pig. In the meantime, if you’d like to learn more about Hadoop, check out our new course.

Apache Hadoop, Hadoop, Apache, the Apache feather logo, and the Apache Hadoop project logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and other countries.

About the Author

Martin Schaeferle is the Vice President of Technology for LearnNowOnline. Martin joined the company in 1994 and started teaching IT professionals nationwide to develop applications using Visual Studio and Microsoft SQL Server. He has been a featured speaker at various conferences including Microsoft Tech-Ed, DevConnections and the Microsoft NCD Channel Summit. Today, he is responsible for all product and software development as well as managing the company’s IT infrastructure. Martin enjoys staying on the cutting edge of technology and guiding the company to produce the best learning content with the best user experience in the industry. In his spare time, Martin enjoys golf, fishing, and being with his wife and three teenage children.

This entry was posted in Uncategorized and tagged big data, Hadoop, Hadoop architecture, Hadoop Distributed File System, HDFS, MapReduce, Martin Schaeferle on July 3, 2014 by Marty S..

SEARCH

Search for: Search

FOLLOW US

LATEST POSTS

- Power Pivot Dashboards
- Hadoop…Pigs, Hives, and Zookeepers, Oh My!
- Bootstrap Fundamentals with Adam Barney
- Watch “Getting Started with AngularJS”
- Agile/Scrum Essentials for Practitioners

ARCHIVES

- July 2014
- June 2014
- May 2014
- April 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- October 2013
- September 2013
- August 2013
- July 2013
- June 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012
- August 2012
- July 2012

Proudly powered by WordPress

- **WIKIS**
- **QUIZ**
- **SEARCH**

Search

The Full Wiki

Search:

Go

More info on Hadoop

- Wikis
  - Encyclopedia
    - **Architecture**
      - Hadoop Distributed File System
      - Job Tracker and Task Tracker: the MapReduce engine
      - Other applications
    - Prominent users
      - Hadoop at Yahoo!
      - Other users
    - Hadoop on Amazon EC2/S3 services
    - Hadoop with Sun Grid Engine
    - See also
    - References
    - Bibliography
    - External links
  - Related links
  - Related topics
- Quiz
  - Quiz

Related topics

- Nutch
- Apache Geronimo
- Cassandra (database)
- Apache Derby
- Apache Ant
- Apache Solr
- Apache ZooKeeper
- Apache Tomcat
- CouchDB
- Lucene

Hadoop: Wikis

Note: Many of our articles have **direct quotes from sources you can cite, within the Wikipedia article**! This article doesn't yet, but we're working on it! See more info or our list of citable articles.

Related top topics

Nutch

Apache Geronimo

Apache Ant

Apache Solr

Apache Tomcat

CouchDB

Categories: Free software programmed in Java >Free system software > Distributed file systems > Cloud computing> Cloud infrastructure

Encyclopedia

From Wikipedia, the free encyclopedia

Apache Hadoop	
Developer(s)	Apache Software Foundation
Stable release	0.20.0 / 2009-04-22; 8 months ago
Written in	Java
Operating system	Cross-platform
Development status	Active
Type	Distributed File System
License	Apache License 2.0
Website	http://hadoop.apache.org/

Apache Hadoop is a Java software framework that supports data-intensive distributed applications under a free license.[1] It enables applications to work with thousands of nodes and petabytes of data. Hadoop was inspired by Google's MapReduce and Google File System(GFS) papers.

Hadoop is a top-level Apache project, being built and used by a community of contributors from all over the world.[2] Yahoo! has been the largest contributor[3] to the project and uses Hadoop extensively in its web search and advertising businesses.[4] IBM and Google have announced a major initiative to use Hadoop to support university courses in distributed computer programming.[5]

Hadoop was created by Doug Cutting (now a Cloudera employee)[6], who named it after his child's stuffed elephant. It was originally developed to support distribution for the Nutch search engine project.[7]

Contents

Architecture

Hadoop consists of the *Hadoop Core*, which provides access to the filesystems that Hadoop supports. "Rack awareness" is an optimization which takes into account the geographic clustering of servers; network traffic between servers in different geographic clusters is minimized.[8]As of June 2008, the list of supported filesystems includes:

- HDFS: Hadoop's own filesystem. This is designed to scale to petabytes of storage and runs on top of the filesystems of the underlying operating systems.
- Amazon S3 filesystem. This is targeted at clusters hosted on the Amazon Elastic Compute Cloud server-on-demand infrastructure. There is no rack-awareness in this filesystem, as it is all remote.
- CloudStore (previously Kosmos Distributed File System) - like HDFS, this is rack-aware.
- FTP Filesystem: this stores all its data on remotely accessible FTP servers.
- Read-only HTTP and HTTPS file systems.

Hadoop Distributed File System

The HDFS filesystem stores large files (an ideal file size is a multiple of 64 MB[9]), across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts. With the default replication value, 3, data is stored on three nodes: two on the same rack, and one on a different rack.

The filesystem is built from a cluster of *data nodes*, each of which serves up blocks of data over the network using a block protocol specific to HDFS. They also serve the data over HTTP, allowing access to all content from a web browser or other client. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high.

A filesystem requires one unique server, the *name node*. This is a single point of failure for an HDFS installation. If the name node goes down, the filesystem is offline. When it comes back up, the name node must replay all outstanding operations. This replay process can take over half an hour for a big cluster.[10] The filesystem includes what is called a *Secondary Namenode*, which misleads some people into thinking that when the primary Namenode goes offline, the Secondary Namenode takes over. In fact, the Secondary Namenode regularly connects with the namenode and downloads a snapshot of the primary Namenode's directory information, which is then saved to a directory. This Secondary Namenode is used together with the edit log of the Primary Namenode to create an up-to-date directory structure.

Another limitation of HDFS is that it cannot be directly mounted by an existing operating system. Getting data into and out of the HDFS file system, an action that often needs to be performed before and after executing a job, can be inconvenient. A Filesystem in Userspace has been developed to address this problem, at least for Linux and some other Unix systems.

Replicating data three times is costly. To alleviate this cost, recent versions of HDFS have erasure coding support whereby multiple blocks of the same file are combined together to generate a parity block. HDFS creates parity blocks asynchronously and then decreases the replication factor of the file from 3 to 2. Studies have shown that this technique decreases the physical storage requirements from a factor of 3 to a factor of around 2.2.

Job Tracker and Task Tracker: the MapReduce engine

Above the file systems comes the MapReduce engine, which consists of one *Job Tracker*, to which client applications submit MapReduce jobs. The Job Tracker pushes work out to available *Task Tracker* nodes in the cluster, striving to keep the work as close to the data as possible. With a rack-aware filesystem, the Job Tracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network. If a Task Tracker fails or times out, that part of the job is rescheduled. If the Job Tracker fails, all ongoing work is lost.

Hadoop version 0.21 adds some checkpointing to this process; the Job Tracker records what it is up to in the filesystem. When a Job Tracker starts up, it looks for any such data, so that it can restart work from where it left off. In earlier versions of Hadoop, all active work was lost when a Job Tracker restarted.

Known limitations of this approach are:

- The allocation of work to task trackers is very simple. Every task tracker has a number of available *slots* (such as "4 slots"). Every active map or reduce task takes up one slot. The Job Tracker allocates work to the tracker nearest to the data with an available slot. There is no consideration of the current active load of the allocated machine, and hence its actual availability.
- If one task tracker is very slow, it can delay the entire MapReduce operation - especially towards the end of a job, where everything can end up waiting for a single slow task. With speculative-execution enabled, however, a single task can be executed on multiple slave nodes.

Other applications

The HDFS filesystem is not restricted to MapReduce jobs. It can be used for other applications, many of which are under way at Apache. The list includes the HBase database, the Apache Mahout machine learning system, and matrix operations. Hadoop can in theory be used for any sort of work that is batch-oriented rather than real-time, very data-intensive, and able to work on pieces of the data in parallel.

Prominent users

Hadoop at Yahoo!

On February 19, 2008, Yahoo! launched what it claimed was the world's largest Hadoop production application. The Yahoo! Search Webmap is a Hadoop application that runs on a more than 10,000 core Linux cluster and produces data that is now used in every Yahoo! Web search query.[11]

There are multiple Hadoop clusters at Yahoo!, each occupying a single datacenter (or fraction thereof). No HDFS filesystems or MapReduce jobs are split across multiple datacenters; instead each datacenter has a separate filesystem and workload. The cluster servers run Linux, and are configured on boot using Kickstart. Every machine bootstraps the Linux image, including the Hadoop distribution. Cluster configuration is also aided through a program called ZooKeeper. Work that the clusters perform is known to include the index calculations for the Yahoo! search engine.

On June 10, 2009, Yahoo! released its own distribution of Hadoop. [12]

Other users

Besides Yahoo!, many other organizations are using Hadoop to run large distributed computations. Some of them include:[2]

- A9.com
- AOL
- Booz Allen Hamilton
- EHarmony
- Facebook

- Freebase
- Fox Interactive Media
- IBM
- ImageShack
- ISI
- Joost
- Last.fm
- LinkedIn
- Metaweb
- Meebo
- Ning
- Powerset (now part of Microsoft)
- Proteus Technologies
- The New York Times
- Rackspace
- Veoh

Hadoop on Amazon EC2/S3 services

It is possible to run Hadoop on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service(S3)[13]. As an example The New York Times used 100 Amazon EC2 instances and a Hadoop application to process 4TB of raw image TIFF data (stored in S3) into 11 million finished PDFs in the space of 24 hours at a computation cost of about $240 (not including bandwidth).[14]

There is support for the S3 filesystem in Hadoop distributions, and the Hadoop team generates EC2 machine images after every release. From a pure performance perspective, Hadoop on S3/EC2 is inefficient, as the S3 filesystem is remote and delays returning from every write operation until the data are guaranteed to not be lost. This removes the locality advantages of Hadoop, which schedules work near data to save on network load.

On April 2, 2009 Amazon announced the beta release of a new service called Amazon Elastic MapReduce which they describe as "a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data. It utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3)."[15]

Hadoop with Sun Grid Engine

Hadoop can also be used in compute farms and high-performance computing environments. Integration withSun Grid Engine was released, and running Hadoop on Sun Grid (Sun's on-demand utility computing service) is possible. [16] In the initial implementation of the integration, the CPU-time scheduler has no knowledge of the locality of the data. A key feature of the Hadoop Runtime, "do the work in the same server or rack as the data" is therefore lost. During the Sun HPC Software Workshop '09, an improved integration with data-locality awareness was announced. [17]

Sun also has the *Hadoop Live CD* OpenSolaris project, which allows running a fully functional Hadoop cluster using a live CD.[18]

See also

Free software portal

- Nutch - an effort to build an open source search engine based on Lucene and Hadoop. Also created by Doug Cutting.
- HBase - BigTable-model database. Sub-project of Hadoop.
- Hypertable - HBase alternative
- MapReduce - Hadoop's fundamental data filtering algorithm
- Apache Pig - Reporting query language for Hadoop
- Apache Mahout - Machine Learning algorithms implement on Hadoop
- Cloud computing

References

1. ^ "Hadoop is a framework for running applications on large clusters of commodity hardware. The Hadoop framework transparently provides applications both reliability and data motion. Hadoop implements a computational paradigm named map/reduce, where the application is divided into many small fragments of work, each of which may be executed or reexecuted on any node in the cluster. In addition, it provides a distributed file system that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. Both map/reduce and the distributed file system are designed so that node failures are automatically handled by the framework." Hadoop Overview
2. ^ *a* *b* Applications and organizations using Hadoop
3. ^ Hadoop Credits Page
4. ^ Yahoo! Launches World's Largest Hadoop Production Application
5. ^ Google Press Center: Google and IBM Announce University Initiative to Address Internet-Scale Computing Challenges
6. ^ Hadoop creator goes to Cloudera
7. ^ "Hadoop contains the distributed computing platform that was formerly a part of Nutch. This includes the Hadoop Distributed Filesystem (HDFS) and an implementation of map/reduce." About Hadoop
8. ^ http://hadoop.apache.org/core/docs/r0.17.2/hdfs_user_guide.html#Rack+Awareness
9. ^ The Hadoop Distributed File System: Architecture and Design

10. ^ Improve Namenode startup performance. "Default scenario for 20 million files with the max Java heap size set to 14GB : 40 minutes. Tuning various Java options such as young size, parallel garbage collection, initial Java heap size : 14 minutes"
11. ^ Yahoo! Launches World's Largest Hadoop Production Application (Hadoop and Distributed Computing at Yahoo!)
12. ^ Hadoop and Distributed Computing at Yahoo!
13. ^ http://aws.typepad.com/aws/2008/02/taking-massive.html Running Hadoop on Amazon EC2/S3
14. ^ Self-service, Prorated Super Computing Fun! - Open - Code - New York Times Blog
15. ^ Amazon Elastic MapReduce Beta
16. ^ "Creating Hadoop pe under SGE". Sun Microsystems. 2008-01-16.
17. ^ "HDFS-Aware Scheduling With Grid Engine". Sun Microsystems. 2009-09-10.
18. ^ "OpenSolaris Project: Hadoop Live CD". Sun Microsystems. 2008-08-29.

Bibliography

- Chuck, Lam (January 28, 2010), *Hadoop in Action* (1st ed.), Manning, pp. 325, ISBN 1935182196
- Venner, Jason (June 22, 2009), *Pro Hadoop* (1st ed.), Apress, pp. 440, ISBN 1430219424
- White, Tom (June 16, 2009), *Hadoop: The Definitive Guide* (1st ed.), O'Reilly Media, pp. 524, ISBN 0596521979

External links

- Official web site

Top level projects	ActiveMQ · Ant · Apache HTTP Server · APR · Beehive · Buildr · Camel · Cayenne ·Cocoon · CouchDB · CXF · Derby · Direc pestry · Tomcat · Tuscany · Velocity · Wicket · XMLBeans
Jakarta Projects	BCEL · BSF · Cactus · ECS · JCS · JMeter · ORO · Regexp
Commons Projects	Sanselan
Lucene Projects	Lucene Java · Droids · Lucene.Net · Lucy · Mahout · Nutch · Open Relevance Project ·PyLu
Other projects	Chainsaw · HBase · Xerces · Batik · FOP · Log4j · XAP · River · ServiceMix · Log4Net ·Abde
Incubator Projects	JSPWiki · Click · Cassandra
Retired projects	HiveMind · Slide · Shale

Categories: Free software programmed in Java | Free system software | Distributed file systems | Cloud computing | Cloud infrastructure

Related links

Up to date as of November 16, 2009

- Mention of Nutch and Hadoop - How Google Works - Infrastructure
- IBM MapReduce Tools - alphaWorks : IBM MapReduce Tools for Eclipse : Overview
- Heritrix Hadoop DFS Writer Processor - The Lab - Zvents
- Hadoop website - Welcome to Apache Hadoop Core!
- A NYT blog - Self-service, Prorated Super Computing Fun! - Open Blog - NYTimes.com
- Hadoop wiki
- Yahoo's bet on Hadoop - Yahoo!'s bet on Hadoop - O'Reilly Radar
- Google Press Center: Google and IBM Announce University Initiative to Address Internet-Scale Computing Challenges - Google Press Center: Press Release
- Yahoo's Doug Cutting on MapReduce and the Future of Hadoop - InfoQ: Yahoo's Doug Cutting on MapReduce and the Future of Hadoop
- Hadoop Overview - ProjectDescription - Hadoop Wiki
- Hadoop wiki
- Hadoop website - Welcome to Apache Hadoop!
- Yahoo! Launches World's Largest Hadoop Production Application - Yahoo! Launches World's Largest Hadoop Production Application (Hadoop and Distributed Computing at Yahoo!)
- Running Hadoop on Amazon EC2/S3 - Amazon Web Services Blog: Taking Massive Distributed Computing to the Common Man - Hadoop on Amazon EC2/S3

Related topics

Up to date as of August 19, 2010

- Nutch
- Apache Geronimo
- Cassandra (database)
- Apache Derby
- Apache Ant
- Apache Solr
- Apache ZooKeeper
- Apache Tomcat
- CouchDB
- Lucene

Got something to say? Make a comment.

Your name

Your email address

Message

Privacy & Terms

Submit Comment »

Version 0609, w