Unit 2. (BigData Analytics)

HDFS (Hadoop Distributed File System)

History of Hadoop

2002

It all started in the year 2002 with the Apache Nutch project.

In 2002, Doug Cutting and Mike Cafarella were working on Apache Nutch Project that aimed at building a web search engine that would crawl and index websites.

After a lot of research, Mike Cafarella and Doug Cutting estimated that it would cost around $500,000 in hardware with a monthly running cost of $30,000 for a system supporting a one-billion-page index.

This project proved to be too expensive and thus found infeasible for indexing billions of webpages. So they were looking for a feasible solution that would reduce the cost.

2003

Meanwhile, In 2003 Google released a search paper on Google distributed File System (GFS) that described the architecture for GFS that provided an idea for storing large datasets in a distributed environment. This paper solved the problem of storing huge files generated as a part of the web crawl and indexing process. But this is half of a solution to their problem.

2004

In 2004, Nutch's developers set about writing an open-source implementation, the Nutch Distributed File System (NDFS).

In 2004, Google introduced MapReduce to the world by releasing a paper on MapReduce. This paper provided the solution for processing those large datasets. It gave a full solution to the Nutch developers.

Google provided the idea for distributed storage and MapReduce. Nutch developers implemented MapReduce in the middle of 2004.

2006

The Apache community realized that the implementation of MapReduce and NDFS could be used for other tasks as well. In February 2006, they came out of Nutch and formed an independent subproject of Lucene called "**Hadoop**" (which is the name of Doug's kid's yellow elephant).

As the Nutch project was limited to 20 to 40 nodes cluster, Doug Cutting in 2006 itself joined Yahoo to scale the Hadoop project to thousands of nodes cluster.

2007

In 2007, Yahoo started using Hadoop on 1000 nodes cluster.

2008

In January 2008, Hadoop confirmed its success by becoming the top-level project at Apache.

By this time, many other companies like Last.fm, Facebook, and the New York Times started using Hadoop.

Various Release of Hadoop

2011 – 2012
On 27 December 2011, Apache released Hadoop version 1.0 that includes support for Security, Hbase, etc.

On 10 March 2012, release 1.0.1 was available. This is a bug fix release for version 1.0.

On 23 May 2012, the Hadoop 2.0.0-alpha version was released. This release contains YARN.

The second (alpha) version in the Hadoop-2.x series with a more stable version of YARN was released on 9 October 2012.

2017 – now
On 13 December 2017, release 3.0.0 was available

On 25 March 2018, Apache released Hadoop 3.0.1, which contains 49 bug fixes in Hadoop 3.0.0.

On 6 April 2018, Hadoop release 3.1.0 came that contains 768 bug fixes, improvements, and enhancements since 3.0.0.

Later, in May 2018, Hadoop 3.0.3 was released.

On 8 August 2018, Apache 3.1.1 was released.

What is Apache Hadoop

Apache Hadoop software is an open source framework that allows for the distributed storage and processing of large datasets across clusters of computers using simple programming models. Hadoop is designed to scale up from a single computer to thousands of clustered computers, with each machine offering local computation and storage. In this way, Hadoop can efficiently store and process large datasets ranging in size from gigabytes to petabytes of data.

Hadoop is based on a few key components:

Hadoop Distributed File System (HDFS): This is the storage component of Hadoop, which distributes and replicates data across multiple nodes in a cluster. It allows for the reliable and scalable storage of large datasets.

MapReduce: This is the processing component of Hadoop. It allows developers to write programs that process and analyze large datasets in parallel across the entire Hadoop cluster.

Hadoop is particularly useful for handling big data applications where traditional databases might struggle due to the sheer volume of data. It enables the processing of data in parallel on a large scale, making it suitable for tasks like data analysis, machine learning, and other data-intensive applications.

Analysing Data with Unix tools

To understand how to work with Unix, data – Weather Dataset is used.
Weather sensors gather information consistently at numerous areas over the globe and assemble an enormous volume of log information, which is a decent possibility for investigation with MapReduce in light of the fact that is required to process every one of the information, and the information is record-oriented and semi-organized.

The information utilized is from the National Climatic Data Center, or NCDC. The information is put away utilizing a line-arranged ASCII group, in which each line is a record. The organization underpins a rich arrangement of meteorological components, huge numbers of which are discretionary or with variable information lengths. For straightforwardness, centre around the fundamental components, for example, temperature, which is constantly present and are of fixed width.

Structure of NCDC record

0057

332130 # USAF weather station identifier

99999 # WBAN weather station identifier

19500101 # observation date

0300 # observation time

4

+51317 # latitude (degrees x 1000)

+028783 # longitude (degrees x 1000)

FM-12

+0171 # elevation (meters)

99999

V020

320 # wind direction (degrees)

1 # quality code

N 0072

1 00450 # sky ceiling height (meters)

1 # quality code

C

N

010000 # visibility distance (meters)

1 # quality code

N

9

-0128 # air temperature (degrees Celsius x 10)

1 # quality code

-0139 # dew point temperature (degrees Celsius x 10)

1 # quality code

10268 # atmospheric pressure (hectopascals x 10)

1 # quality code

Analyzing Data with Hadoop

Analyzing data with Hadoop involves using various components and tools within the Hadoop ecosystem to process, transform, and gain insights from large datasets. Here are the steps and considerations for analyzing data with Hadoop:

1. Data Ingestion:

- Start by ingesting data into the Hadoop cluster. You can use tools like Apache Flume, Apache Kafka, or Hadoop's HDFS for batch data ingestion.
- Ensure that your data is stored in a structured format in HDFS or another suitable storage system.

2. Data Preparation:

- Preprocess and clean the data as needed. This may involve tasks such as data deduplication, data normalization, and handling missing values.
- Transform the data into a format suitable for analysis, which could include data enrichment and feature engineering.

3. Choose a Processing Framework:

- Select the appropriate data processing framework based on your requirements. Common choices include:
  - MapReduce: Ideal for batch processing and simple transformations.
  - Apache Spark: Suitable for batch, real-time, and iterative data processing. It offers a wide range of libraries for machine learning, graph processing, and more.
  - Apache Hive: If you prefer SQL-like querying, you can use Hive for data analysis.
  - Apache Pig: A high-level data flow language for ETL and data analysis tasks.
  - Custom Code: You can write custom Java, Scala, or Python code using Hadoop APIs if necessary.

4. Data Analysis:

- Write the code or queries needed to perform the desired analysis. Depending on your choice of framework, this may involve writing MapReduce jobs, Spark applications, HiveQL queries, or Pig scripts.
- Implement data aggregation, filtering, grouping, and any other required transformations.

5. Scaling:

- Hadoop is designed for horizontal scalability. As your data and processing needs grow, you can add more nodes to your cluster to handle larger workloads.

6. Optimization:

- Optimize your code and queries for performance. Tune the configuration parameters of your Hadoop cluster, such as memory settings and resource allocation.
- Consider using data partitioning and bucketing techniques to improve query performance.

7. Data Visualization:

- Once you have obtained results from your analysis, you can use data visualization tools like Apache Zeppelin, Apache Superset, or external tools like Tableau and Power BI to create meaningful visualizations and reports.

8. Iteration:

- Data analysis is often an iterative process. You may need to refine your analysis based on initial findings or additional questions that arise.

9. Data Security and Governance:

- Ensure that data access and processing adhere to security and governance policies. Use tools like Apache Ranger or Apache Sentry for access control and auditing.

10. Results Interpretation:

- Interpret the results of your analysis and draw meaningful insights from the data.
- Document and share your findings with relevant stakeholders.

11. Automation:

- Consider automating your data analysis pipeline to ensure that new data is continuously ingested, processed, and analyzed as it arrives.

12. Performance Monitoring:

- Implement monitoring and logging to keep track of the health and performance of your Hadoop cluster and data analysis jobs.
-

The Design of HDFS,HDFS Concepts

With growing data velocity the data size easily outgrows the storage limit of a machine. A solution would be to store the data across a network of machines. Such filesystems are called distributed filesystems. Since data is stored across a network all the complications of a network come in.

This is where Hadoop comes in. It provides one of the most reliable filesystems. HDFS (Hadoop Distributed File System) is a unique design that provides storage for extremely large files with streaming data access pattern and it runs on commodity hardware. Let’s elaborate the terms:

Extremely large files: Here we are talking about the data in range of petabytes(1000 TB).

Streaming Data Access Pattern: HDFS is designed on principle of write-once and read-many-times. Once data is written large portions of dataset can be processed any number times.

Commodity hardware: Hardware that is inexpensive and easily available in the market. This is one of feature which specially distinguishes HDFS from other file system.

Nodes: Master-slave nodes typically forms the HDFS cluster.

NameNode(MasterNode):

Manages all the slave nodes and assign work to them.

It executes filesystem namespace operations like opening, closing, renaming files and directories.

It should be deployed on reliable hardware which has the high config. not on commodity hardware.

DataNode(SlaveNode):

Actual worker nodes, who do the actual work like reading, writing, processing etc.

They also perform creation, deletion, and replication upon instruction from the master.

They can be deployed on commodity hardware.

HDFS daemons: Daemons are the processes running in background.

Namenodes:

Run on the master node.

Store metadata (data about data) like file path, the number of blocks, block Ids. etc.

Require high amount of RAM.

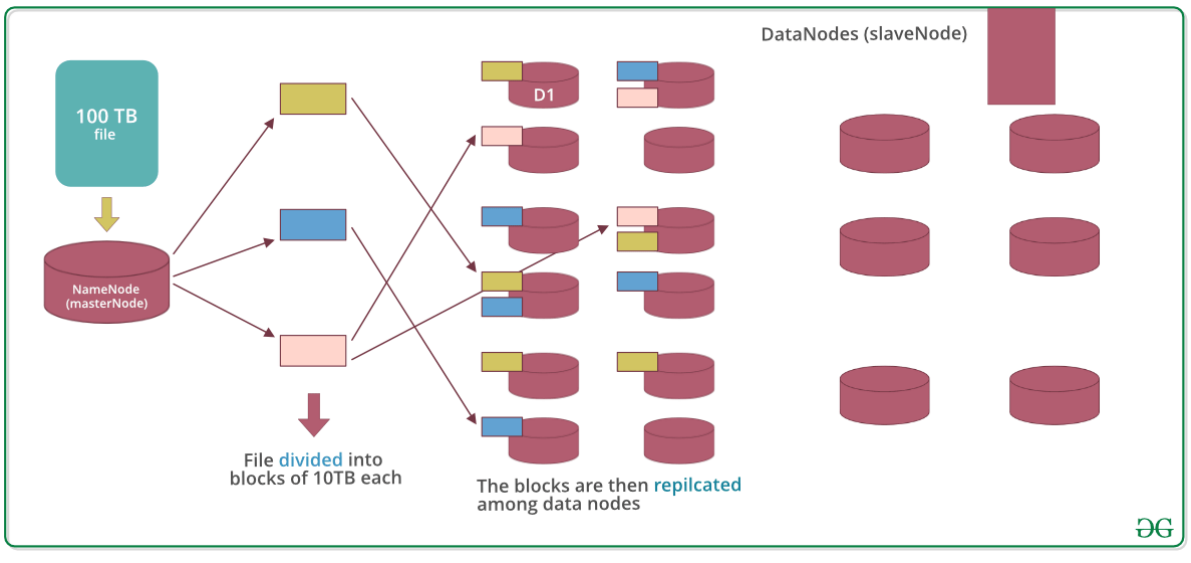
Store meta-data in RAM for fast retrieval i.e to reduce seek time. Though a persistent copy of it is kept on disk.

DataNodes:

Run on slave nodes.

Require high memory as data is actually stored here.

Data storage in HDFS: Now let’s see how the data is stored in a distributed manner.



Lets assume that 100TB file is inserted, then masternode(namenode) will first divide the file into blocks of 10TB (default size is 128 MB in Hadoop 2.x and above). Then these blocks are stored across different datanodes(slavenode). Datanodes(slavenode)replicate the blocks among themselves and the information of what blocks they contain is sent to the master. Default replication factor is 3 means for each block 3 replicas are created (including itself). In hdfs.site.xml we can increase or decrease the replication factor i.e we can edit its configuration here.

- **Command Line Interface**

To interact with Hadoop Distributed File System (HDFS) from the command line, you can use the Hadoop command-line interface (CLI) tools. Here are some commonly used commands:

1. **Copying files to HDFS:**

   bashCopy code

   > hdfs dfs -copyFromLocal <local-source> <hdfs-destination>

   Example:

   bashCopy code

   hdfs dfs -copyFromLocal /local/path/file.txt /user/hadoop/hdfs/path/

2. **Copying files from HDFS to the local file system:**

   bashCopy code

   hdfs dfs -copyToLocal <hdfs-source> <local-destination>

   Example:

   bashCopy code

   hdfs dfs -copyToLocal /user/hadoop/hdfs/path/file.txt /local/path/

3. **Listing files in a directory:**

   bashCopy code

   hdfs dfs -ls <hdfs-directory>

   Example:

   bashCopy code

   hdfs dfs -ls /user/hadoop/hdfs/path/

4. **Creating a directory in HDFS:**

   bashCopy code

   hdfs dfs -mkdir <hdfs-directory>

   Example:

   bashCopy code

   hdfs dfs -mkdir /user/hadoop/new_directory

5. **Removing a file or directory from HDFS:**

   bashCopy code

   hdfs dfs -rm <hdfs-path>

   Example:

   bashCopy code

   hdfs dfs -rm /user/hadoop/hdfs/path/file.txt

6. **Moving or renaming a file or directory in HDFS:**

bashCopy code

```
hdfs dfs -mv <source> <destination>
```

Example:

bashCopy code

```
hdfs dfs -mv /user/hadoop/old_directory /user/hadoop/new_directory
```

7. **Displaying the contents of a file:**

bashCopy code

```
hdfs dfs -cat <hdfs-file>
```

Example:

bashCopy code

```
hdfs dfs -cat /user/hadoop/hdfs/path/file.txt
```

These commands assume that you have the Hadoop command-line tools installed and configured. Additionally, replace placeholders like **<local-source>**, **<hdfs-destination>**, etc., with actual paths or filenames based on your environment and requirements.

- **Hadoop file system interfaces**

Hadoop is capable of running various file systems and HDFS is just one single implementation that out of all those file systems. The Hadoop has a variety of file systems that can be implemented concretely. The Java abstract class ***org.apache.hadoop.fs.FileSystem*** represents a file system in Hadoop.

Filesystem	**URI scheme**	**Java implementation (all under org.apache.hadoop)**	**Description**
Local	file	fs.LocalFileSystem	The Hadoop Local filesystem is used for a locally connected disk with client-side checksumming. The local filesystem uses RawLocalFileSystem with no checksums.
HDFS	hdfs	hdfs.DistributedFileSystem	HDFS stands for Hadoop Distributed File System and it is drafted for working with MapReduce efficiently.
HFTP	hftp	hdfs.HftpFileSystem	The HFTP filesystem provides read-only access to HDFS over HTTP. There is no connection of HFTP with FTP. This filesystem is commonly used with ***distcp*** to share data between

Filesystem	URI scheme	Java implementation (all under org.apache.hadoop)	Description
			HDFS clusters possessing different versions.
HSFTP	hsftp	hdfs.HsftpFileSystem	The HSFTP filesystem provides read-only access to HDFS over HTTPS. This file system also does not have any connection with FTP.
HAR	har	fs.HarFileSystem	The HAR file system is mainly used to reduce the memory usage of NameNode by registering files in Hadoop HDFS. This file system is layered on some other file system for archiving purposes.
KFS (Cloud-Store)	kfs	fs.kfs.KosmosFileSystem	cloud store or KFS(KosmosFileSystem) is a file system that is written in c++. It is very much similar to a distributed file system like HDFS and GFS(Google File System).
FTP	ftp	fs.ftp.FTPFileSystem	The FTP filesystem is supported by the FTP server.
S3 (native)	s3n	fs.s3native.NativeS3FileSystem	This file system is backed by AmazonS3.
S3 (block-based)	s3	fs.s3.S3FileSystem	S3 (block-based) file system which is supported by Amazon s3 stores files in blocks(similar to HDFS) just to overcome S3's file system 5 GB file size limit.

Hadoop gives numerous interfaces to its various filesystems, and it for the most part utilizes the URI plan to pick the right filesystem example to speak with. You can use any of this filesystem for working with MapReduce while processing very large datasets but distributed file systems with data locality features are preferable like HDFS and KFS(KosmosFileSystem).

- **Hadoop's basic data flow**
-

A basic data flow of the Hadoop system can be divided into four phases:

1. **Capture Big Data** : The sources can be extensive lists that are structured, semi-structured, and unstructured, some streaming, real-time data sources, sensors, devices, machine-captured data, and many other sources. For data capturing and storage, we have different data integrators such as, Flume, Sqoop, Storm, and so on in the Hadoop ecosystem, depending on the type of data.
2. **Process and Structure**: We will be cleansing, filtering, and transforming the data by using a MapReduce-based framework or some other frameworks which can perform distributed programming in the Hadoop ecosystem. The frameworks available currently are MapReduce, Hive, Pig, Spark and so on.
3. **Distribute Results**: The processed data can be used by the BI and analytics system or the big data analytics system for performing analysis or visualization.
4. **Feedback and Retain**: The data analyzed can be fed back to Hadoop and used for improvements...

- **Data Ingest with Flume and Scoop and Hadoop archives**

- **What is Sqoop in Hadoop?**

Apache Sqoop (SQL-to-Hadoop) is a lifesaver for anyone who is experiencing difficulties in moving data from the data warehouse into the Hadoop environment. Apache Sqoop is an effective hadoop tool used for importing data from RDBMS's like MySQL, Oracle, etc. into HBase, Hive or HDFS. Sqoop hadoop can also be used for exporting data from HDFS into RDBMS. Apache Sqoop is a command line interpreter i.e. the Sqoop commands are executed one at a time by the interpreter.

Need for Apache Sqoop

With increasing number of business organizations adopting Hadoop to analyse huge amounts of structured or unstructured data, there is a need for them to transfer petabytes or exabytes of data between their existing relational databases, data sources, data warehouses and the Hadoop environment. Accessing huge amounts of unstructured data directly from MapReduce applications running on large Hadoop clusters or loading it from production systems is a complex task because data transfer using scripts is often not effective and time consuming.

How Apache Sqoop works?

Sqoop is an effective hadoop tool for non-programmers which functions by looking at the databases that need to be imported and choosing a relevant import function for the source data. Once the input is recognized by Sqoop hadoop, the metadata for the table is read and a class definition is created for the input requirements. Hadoop Sqoop can be forced to function selectively by just getting the columns needed before input instead of importing the entire input and looking for the data in it. This saves considerable amount of time. In reality, the import from the database to HDFS is accomplished by a MapReduce job that is created in the background by Apache Sqoop.

WHAT IS FLUME IN HADOOP?

Apache Flume is service designed for streaming logs into Hadoop environment. Flume is a distributed and reliable service for collecting and aggregating huge amounts of log data. With

a simple and easy to use architecture based on streaming data flows, it also has tunable reliability mechanisms and several recovery and failover mechanisms.

Need for Flume

Logs are usually a source of stress and argument in most of the big data companies. Logs are one of the most painful resources to manage for the operations team as they take up huge amount of space. Logs are rarely present at places on the disk where someone in the company can make effective use of them or hadoop developers can access them. Many big data companies wind up building tools and processes to collect logs from application servers, transfer them to some repository so that they can control the lifecycle without consuming unnecessary disk space.

This frustrates developers as the logs are often not present at the location where they can view them easily, they have limited number of tools available for processing logs and have confined capabilities in intelligently managing the lifecycle. Apache Flume is designed to address the difficulties of both operations group and developers by providing them an easy to use tool that can push logs from bunch of applications servers to various repositories via a highly configurable agent.

How Apache Flume works?

Flume has a simple event driven pipeline architecture with 3 important roles-Source, Channel and Sink.

- Source defines where the data is coming from, for instance a message queue or a file.
- Sinks defined the destination of the data pipelined from various sources.
- Channels are pipes which establish connect between sources and sinks.

Apache flume works on two important concepts-

1. The master acts like a reliable configuration service which is used by nodes for retrieving their configuration.
2. If the configuration for a particular node changes on the master then it will dynamically be updated by the master.

Node is generally an event pipe in Hadoop Flume which reads from the source and writes to the Sink. The characteristics and role of a flume node is determine by the behaviour of source and sinks. Apache Flume is built with several source and sink options but if none of them fits in your requirements then developers can write their own. A flume node can also be configured with the help of a sink decorator which can interpret the event and transforms it as it passes through. With all these basic primitives, developers can create different topologies to collect data on any application server and direct it to any log repository.

Hadoop archives

Hadoop archives are special format archives. A Hadoop archive maps to a file system directory. A Hadoop archive always has a *. har extension. A Hadoop archive directory contains metadata (in the form of _index and _masterindex) and data (part-*) files.

A Hadoop Archive, often referred to as HAR, is a file archive format used in Hadoop to store and manage a large number of small files efficiently. It is designed to address the challenges associated with storing and processing a vast number of small files in Hadoop's distributed file system, HDFS (Hadoop Distributed File System).

Here are the key characteristics and purposes of Hadoop Archives (HARs):

1. **File Consolidation**: HAR files consolidate a large number of small files into a single archive file. This consolidation helps reduce the overhead associated with managing metadata for each small file in HDFS.

2. **Metadata Reduction**: Hadoop's NameNode stores metadata information for each file and directory in HDFS. When there are a massive number of small files, this metadata can become a significant overhead. HARs reduce this overhead by grouping multiple files into one.

3. **Compression**: HAR files can be compressed, which further reduces storage space and improves data transfer efficiency.

4. **Efficient Processing**: When it comes to processing small files, such as during MapReduce jobs, HARs can significantly improve job performance by reducing the number of file operations and speeding up data access.

- **Hadoop I/O: Compression**

1) Compressing input files

If the input file is compressed, then the bytes read in from HDFS is reduced, which means less time to read data. This time conservation is beneficial to the performance of job execution.

If the input files are compressed, they will be decompressed automatically as they are read by MapReduce, using the filename extension to determine which codec to use. For example, a file ending in .gz can be identified as gzip-compressed file and thus read with GzipCodec.

2) Compressing output files

Often we need to store the output as history files. If the amount of output per day is extensive, and we often need to store history results for future use, then these accumulated results will take extensive amount of HDFS space. However, these history files may not be used very frequently, resulting in a waste of HDFS space. Therefore, it is necessary to compress the output before storing on HDFS.

3) Compressing map output

Even if your MapReduce application reads and writes uncompressed data, it may benefit from compressing the intermediate output of the map phase. Since the map output is written to disk and transferred across the network to the reducer nodes, by using a fast compressor such as LZO or Snappy, you can get performance gains simply because the volume of data to transfer is reduced.

2. Common input format

Compression format	Tool	Algorithm	File extention	Splittable
gzip	*gzip*	DEFLATE	.gz	No
bzip2	*bizp2*	bzip2	.bz2	Yes
LZO	*lzop*	LZO	.lzo	Yes if indexed
Snappy	N/A	Snappy	.snappy	No

What is the Data Serialization Storage format?

The RPC protocol uses serialization to render the message into a binary stream to be sent to the remote node, which then deserializes the binary stream into the original message.

Storage formats are a way to define how to store information in the file. Most of the time, assume this information is from the extension of the data. Both structured and unstructured data can store on HADOOP-enabled systems. Common Hdfs file formats are -

- Plain text storage
- Sequence files
- RC files
- AVRO
- Parquet

Why Storage Formats?

- File format must be handy to serve complex data structures
- HDFS enabled applications to find relevant data in a particular location and write back data to another location.
- Dataset is large
- Having schemas
- Having storage constraints

Avro

What is Avro?

Apache Avro is a language-neutral data serialization system. It was developed by Doug Cutting, the father of Hadoop. Since Hadoop writable classes lack language portability, Avro becomes quite helpful, as it deals with data formats that can be processed by multiple languages. Avro is a preferred tool to serialize data in Hadoop.

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

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

File-Based Data structures

File-based data structures
Two file formats:
1, Sequencefile
2, MapFile

Sequencefile

1. sequencefile files are <key,value>flat files (Flat file) designed by Hadoop to store binary forms of pairs.

2, can sequencefile as a container, all the files packaged into the Sequencefile class can be efficiently stored and processed small files .

3. sequencefile files are not sorted by their stored key, Sequencefile's internal class writer** provides append functionality * *.

4. The key and value in Sequencefile can be any type writable or a custom writable type.

Sequencefile Compression

1. The internal format of the sequencefile depends on whether compression is enabled, or, if it is, either a record compression or a block compression.
2, three kinds of types:
A. No compression type : If compression is not enabled (the default setting), then each record consists of its record length (number of bytes), the length of the key, the key and the value. The Length field is four bytes.
B. Record compression type : The record compression format is basically the same as the uncompressed format, and the difference is that the value byte is compressed with the encoder defined in the header. Note that the key is not compressed.
C. Block compression type : Block compression compresses multiple records at once , so it is more compact than record compression and generally preferred . When the number of bytes recorded reaches the minimum size, it is added to the block. The minimum valueio.seqfile.compress.blocksizeis defined by the property in. The default value is 1000000 bytes. The format is record count, key length, key, value length, value.