ISSN:0975-9646

Review on Data Encryption in Hadoop Distributed File System Using AES Algorithm with Key Management

Vrushali Patil, Rajan Jamgekar, Namdev Sawant

Computer Science and Engineering Department, Solapur University, Solapur, India.

SKN Sinhgad College of Engineering, Korti, Pandharpur, India.

***Abstract*—Hadoop is an java base programming framework that supports the processing and storage of extremely large data sets in a distributed computing environment. Data security is an important issue as far as storage of sensitive data is concerned. Hadoop by default does not contain any security mechanism but as it has grown very much and it is the first choice to store and manage data it is necessary to introduce security solutions to Hadoop in order to secure the important data in the Hadoop environment. Encryption of large amount of data stored in HDFS is actually a process which takes a lot of time and this time consuming process of encryption should be controlled by encrypting the data using a parallel method. This study discusses a new technique to perform encryption in parallel using AES algorithm.**

***Keywords*— Hadoop, Hadoop distributed file systems (HDFS), Data Encryption, MapReduce and AES algorithm**

I. INTRODUCTION

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Hadoop has been developed under an Apache License. Hadoop is a framework of tools which supports running application on big data and it is implemented in Java. Hadoop consists of two main modules: the MapReduce and the Hadoop Distributed File System[6]

Hadoop File System was developed using distributed file system design. It runs on commodity hardware. HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.[3]

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.It is a processing technique and a program model for distributed computing based on java.The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes.[2]

A. *Architecture of HDFS :*

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.[3]

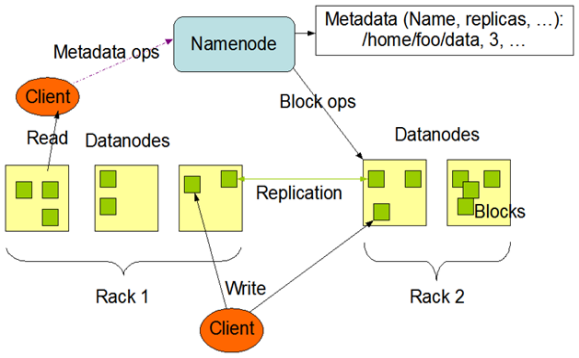

Figure 1: HDFC Architecture[6]

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.

II. BIG DATA HADOOP 'S TRADITIONAL SECURITY

Originally Hadoop was developed without security in mind, no security model, no authentication of users and services and no data privacy, so anybody could submit arbitrary code to be executed. Although auditing and authorization controls (HDFS file permissions and ACLs) were used in earlier distributions, such access control was easily evaded because any user could impersonate any other user. Because impersonation was frequent and done by most users, the security controls measures that did subsist were not very effective. Later authorization and authentication was added, but that to have some weakness in it. Because there were very few security control measures within Hadoop ecosystem, many fortuity and security incidents happened in such environments. Well-meant users can make mistakes (e.g. deleting massive amounts of data within seconds with a distributed delete). All users and programmers had the same level of access privileges to all the data in the cluster, any job could access any of the data in the cluster, and any user could read any data set [4]. Because MapReduce had no concept of authentication or authorization, an impish user could lower the priorities of other Hadoop jobs in order to make his job complete faster or to be executed first – or worse, he could kill the other jobs.[1]

B. *Security Issue* :

Hadoop present security issues for data centre managers and security professionals. The security issues are as below

1. Fragmented Data: Big Data clusters contain data that allow multiple copies moving to-and-fro various nodes ensuring redundancy and resiliency. The data that is available for fragmentation and can be shared across multiple servers more complexity is added as a result of the fragmentation which poses a security issue due to the absence of a security model.
2. Distributed Computing: the data source is not fixed resources are processed where available, these lead to large levels of parallel computation. Complicated environments are created that are at high risks of attacks than their counterparts of repositories that are centrally managed and monolithic.
3. Controlling Data Access: big data only provides access control at schema level. There is no finer granularity in addressing proposed users in terms of roles and access related scenarios.
4. Node-to-node communication: Hadoop don't implement secure communication; they use the RPC (Remote Procedure Call) over TCP/IP.
5. Client Interaction: Communication of client takes place with resource manager, data nodes. Clients that have been compromised tend to propagate malicious data or links to either service.
6. Virtually no security: big data stacks where designed with no security in mind. There is no security for common web threats too.[4]

III. SOLUTION FOR BIG DATA SECURITY IN HADOOP

The security features Hadoop to prevent malicious user impersonation. The Hadoop daemons leverage Kerberos to perform user authentication on all remote procedure calls (RPCs). Group resolution is performed on the Hadoop master nodes, NameNode, JobTracker and ResourceManager to guarantee that group membership cannot be manipulated by users. Map tasks are run under the user account of the user who submitted the job, ensuring isolation there. In addition to these features, new authorization mechanisms have been introduced to HDFS and MapReduce to enable more control over user access to data.[4]

C. *Encryption In HDFS* :

HDFS encryption implements transparent, end-to-end encryption of data read from and written to HDFS blocks across your cluster. *Transparent* means that end-users are unaware of the encryption/decryption processes, and *end-to-end* means that data is encrypted at-rest and in-transit.[12]

a. Background :

Encryption can be done at different layers in a traditional data management software/hardware stack. Choosing to encrypt at a given layer comes with different advantages and disadvantages.

1. Application-level encryption: This is the most secure and most flexible approach. The application has ultimate control over what is encrypted and can precisely reflect the requirements of the user. However, writing applications to do this is hard. This is also not an option for customers of existing applications that do not support encryption.[10]

2. Database-level encryption:Similar to application-level encryption in terms of its properties. Most database vendors offer some form of encryption. However, there can be performance issues. One example is that indexes cannot be encrypted.[10]
3. Filesystem-level encryption. This option offers high performance, application transparency, and is typically easy to deploy. However, it is unable to model some application-level policies. For instance, multi-tenant applications might want to encrypt based on the end user. A database might want different encryption settings for each column stored within a single file.[10]
4. Disk-level encryption. Easy to deploy and high performance, but also quite inflexible. Only really protects against physical theft.
5. HDFS-level encryption fits between database-level and filesystem-level encryption in this stack. This has a lot of positive effects. HDFS encryption is able to provide good performance and existing Hadoop applications are able to run transparently on encrypted data. HDFS also has more context than traditional filesystems when it comes to making policy decisions.HDFS-level encryption also prevents attacks at the filesystem-level and below (so-called "OS-level attacks"). The operating system and disk only interact with encrypted bytes, since the data is already encrypted by HDFS.[10]

b. Architecture :

Figure 2 indicates operation that spare each piece into HDFS, customer split every record into settled size square and scrambles it before transfer to Hadoop document framework. It is accounted for that encryption and unscrambling can be actualized essentially by utilizing Java class [1]. Customers, itself perform encryption utilizing AES calculation on the CPU and exchange encoded piece to HDFS (DataNode). At that point collector DataNode (First DataNode where piece store) reproduce hinder into two different DataNodes.[1]

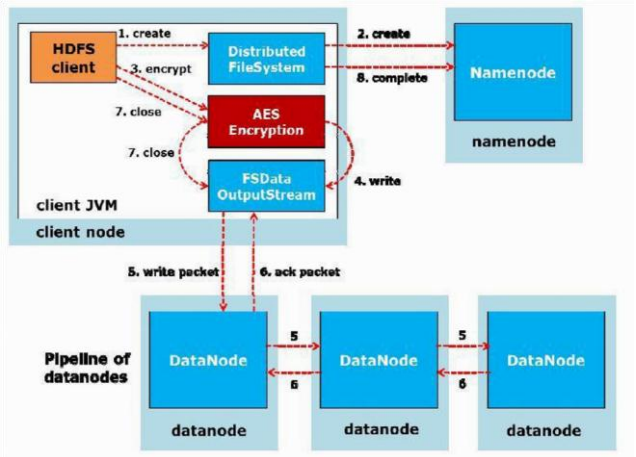

Figure 2: Writing a file by adding an encryption[1]

D. DECRYPTION IN HDFS

Information pieces are composed by customer to DataNode successively, however amid execution of MapReduce occupation numerous squares areperused (decoded) parallel at TaskTracker. Figure 3 demonstrates that MapTask read and encode information obstructs at TaskTracker utilizing AES encryption strategy. It is accounted for that various MapTasks are executing in Hadoop at specialist destinations. HDFS bolsters compose once-read-many model, it is accounted for that simultaneous decoding of HDFS square well reasonable for some MapReduce employments[1]

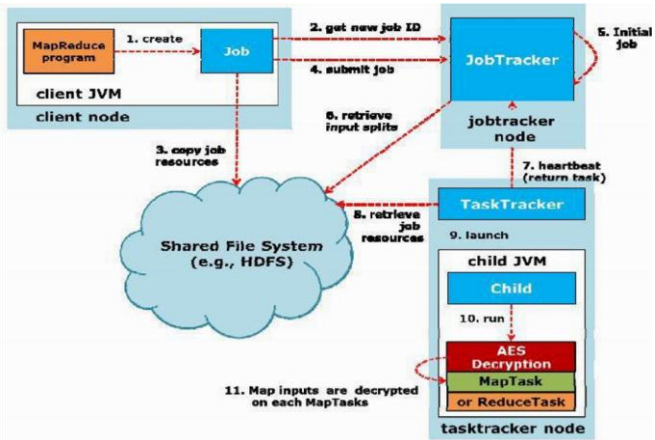

Figure 3: A MapReduce job that read an encrypted file[1]

IV. FUTURE SCOPE

Enormous information contains delicate and private data, so as to secure this huge volume that put away at various product equipment, important to actualize confirmation to check client or framework personality. Approval is valuable for giving access control benefits to client or framework; additionally the ACL's are aides for document consent. OAuth 2.0 is great decision for both validation and Authorization. Furthermore, review trails utilized for following every client action. OAuth 2.0 token effective component that bolster AES to give information classification and uprightness among various client.

V. CONCLUSION

In the period of Big Data, where information is gathered from various sources, security is a measure issue, as there no any settled wellspring of information and no sort of security instrument. Hadoop received by different businesses to process such information, requests solid security arrangement. Consequently verification, approval and encryption or decoding strategies are much supportive to secure Hadoop record framework

REFERENCES

[1] Seonyoung Park and Youngseok Lee ―Secure Hadoop with Encrypted HDFS"
[2] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Cluster. In:OSDI (2004)
[3] Ghemawat, S., Gobioff, H., Leung, S.: The Google File System. In: ACM Symposium onOperating Systems Principles (October 2003)
[4] O'Malley, O., Zhang, K., Radia, S., Marti, R., Harrell, C.: Hadoop Security Design,Technical Report (October 2009)
[5] White, T.: Hadoop: The Definitive Guide, 1st edn. O'Reilly Media (2009)
[6] Hadoop, http://hadoop.apache.org/
[7] Jason Cohen and Dr. Subatra Acharya ―Towards a Trusted Hadoop Storage Platform:Design Considerations of an AES Based Encryption Scheme with TPM Rooted KeyProtections‖ (2013)
[8] Lin, H., Seh, S., Tzeng, W., Lin, B.P. ‖ Toward Data Confidentiality via Integrating sfsfHybrid Encryption Schemes and Hadoop Distributed FileSystem‖ (2012)
[9] Thanh Cuong Nguyen, Wenfeng Shen, Jiwei Jiang and Weimin Xu ―A Novel Data Encryption in HDFS‖ (2013)
[10] Devaraj Das, Owen O'Malley, Sanjay Radia and Kan Zhang ―Adding Security to Apache Hadoop‖
[11] Songchang Jin, Shuqiang Yang, Xiang Zhu, and Hong Yin ―Design of a Trusted File System Based on Hadoop " 2013
[12] Advanced Encryption Standard, http://en.wikipedia.org/wiki/Advanced_ Encryption_Standard [13] Sharma Y. ; Kumar S. and Pai R.M; ―Formal Verification of OAuth 2.0 Using Alloy Framework