Information Flow Control in Hadoop

B.Tech. Project Report

Submitted in partial fulfilment of requirements for the degree of
Bachelor of Technology

by
Karan Ganju
Roll No : 120050021
and
Krishna Deepak
Roll No : 120050057

under the guidance of
Prof. R. K. Shyamasundar

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai 400076, India

December, 2015

Abstract

Discretionary access controls by themselves are not sufficiently able to protect the confidentiality of data. In a situation where multiple users can access the data, trojan attacks can render DAC measures futile. Hence, this calls the need for information flow control as a mandatory access control. The Readers Writers Flow Model is a label model for Information Flow Control that assigns purpose-based labels to subjects and objects and uses these labels to identify potentially malicious data flows. RWFM has been previously realized in the context of File Systems. We explore information flow control in the form of RWFM in Hadoop's map-reduce framework. We believe that a correct labeling of mapper and reducer nodes can help identify possible data breaches. Hence, we aim to create a guard and control monitor that detects and permits only authorized data flows.

Contents

Chapter 1

Introduction

The development of cloud computing has brought about a change in the amount of data that can be collected and processed to obtain meaningful results. With more data being collected by companies to provide better services, data security and privacy are gaining more importance. Privacy breaches are nothing new and the larger the data store, the more the impact. This makes it all the more imperative to employ data security measures for big data solutions like Hadoop.

A particular problem with cloud computing, popularly termed in relevant literature as the multitenant problem, is that many users through their applications simultaneously have access to the same shared resources. This framework facilitates both attacks on confidentiality and integrity of the data. Let us take the case of a server which takes as input map-reduce programs from it's users. Multiple applications submitted by many users will be running simultaneously. An attacker can, due to the sharing of resources, create a situation where his program runs on the same core as the victim's program. Then, he can obtain data related to the victim through many resources such as the processor cache. More coordinated attacks can possibly even result in data breaches.

To prevent such a situation, we need to ensure that history of each node, the level of data that it has touched (read/written) is somehow captured, through labeling or otherwise. This watermark can then be used to identify and prevent potential confidentiality breaches. Hence, there is a need for Information Flow Control in cloud computing systems.

Apache Hadoop, by itself, was developed with little regard to security. There are security add-ons such as for authentication (Apache Knox), authorization (Apache Sentry) and encryption (Clouderas Project Rhino). However, none of these tackle the issue of Information Flow Control. We specifically take up the aim of introducing RWFM in Hadoop.

Chapter 2

Background

2.1 Hadoop Architecture

Hadoop basically consists of 2 components - the Hadoop Distributed File System or HDFS and Hadoop's Yet Another Resource Negotiator or YARN. HDFS takes care of storage while YARN is responsible for coordination of data processing activities. While all data, input, intermediary and output, resides on the HDFS, it is YARN's responsibility to provide computational resources for the applications execution. Based on the idea that *it is easier to move computation than it is to move data*, the HDFS stores data in multiple nodes where applications work on them locally and parallely. Hadoop is generally run on a multi-node setup and follows a master-slave interaction between nodes. The master node consists of the name node, data node, resource manager and node manager while the slave node consists of only the node manager and the data node. These are explained further in the subsections below.

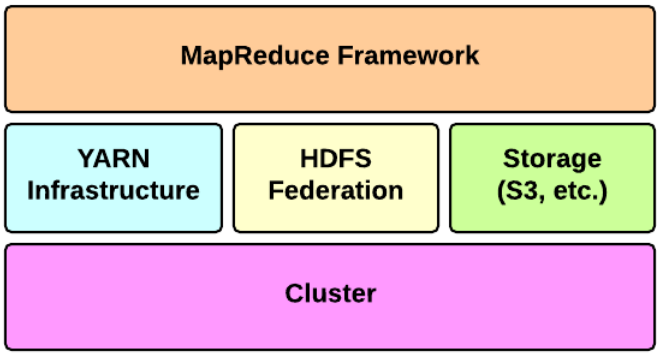

Figure 2.1: Hadoop Overview

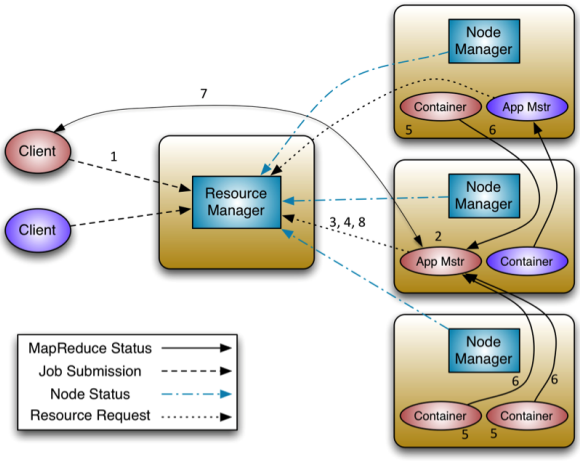

Figure 2.2: Sequence of Events in YARN

2.1.1 YARN

The two important elements of YARN are the node manager which are many per cluster and act as the slaves and the resource manager which is the master of the infrastructure. The resource manager is responsible for the following important tasks apart from others

- Schedule resources appropriately to different tasks
- Check the liveness of the node manager
- Checks task status

The resource manager communicates with live node managers and depending on their offered resources, assigns them tasks. The node managers on the other hand is responsible for sending periodic heartbeats to the resource manager and for offering resources to be utilized to the resource manager.

Each time a client submits a new task, an Application Master is created for the application which negotiates with the Resource Manager for resource allocation. The resource manager, having constant contact with the node managers and an overview of the available resources in all the different nodes, allocates resources to the Application Master appropriately. This is then communicated to the Node managers which reserve the required resources and start the map/combine/reduce task as is required. These events are illustrated in Fig 2.2.

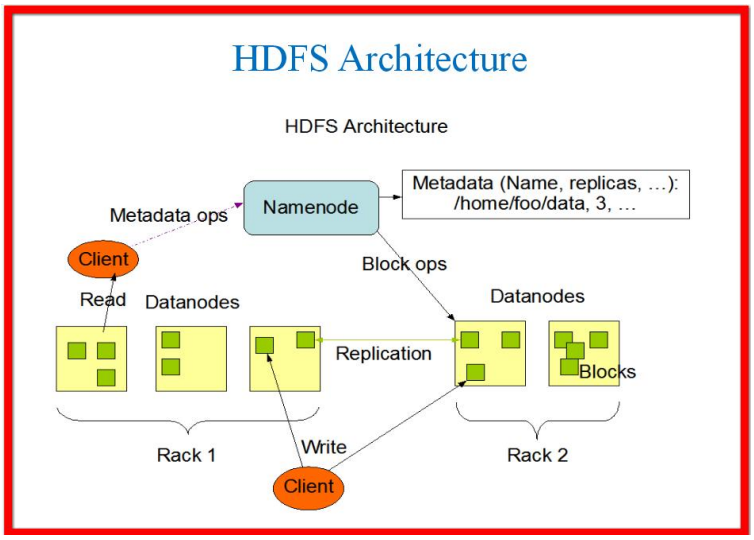

Figure 2.3: Sequence of Events in YARN

2.1.2 HDFS

The Hadoop distributed file system is customized specifically to host large volumes of data. It consists of a single Name Node hosted on the master node which maintains all the metadata of the filesystem and the mapping of data blocks to nodes. Each node on the other hand has a Data Node hosted on it which manages the storage of data on the node. One relevant point to keep in mind is that Hadoop maintains data block replication, i.e., a particular data block may be present on more than one node. This is done to keep the file system failure tolerant.

This architecture of the filesystem leaves a single point of failure, the master node or the Name node, which stores all the metadata. To mitigate for this, another component of Hadoop, called the secondary Name node, periodically creates and stores snapshots of the primary Name node. Figure 2.3 illustrates the interaction between data nodes and name nodes.

2.1.3 Map Reduce Task

It is essential to know of the interactions that happen throughout the map reduce task as these are the points where specific guards will have to be employed to ensure authorized data flows.

As explained previously, Hadoop tries to exploit data locality to reduce IO bandwidth. So

while the Resource Manager allocates resources for the application, it first tries to allocate them on the data node where the relevant data is located. If this cannot happen, due to lack of resources, it tries elsewhere. As soon as these resources, memory and CPU cores, are allotted, the Map task is started.

The map task, after some basic initialization activity, calls the map function that is passed by the user. The key-value pairs that are created are written into an in-memory buffer. A separate spiller thread is invoked as soon as a certain amount of the buffer is filled and it sorts these key-value pairs according to the key and partitions these creating one per reduce task. These partitions are written into the local filesystem, clearing up the in-memory buffer.

Unlike the map task, the reduce task does not exploit data locality. It can be started before the map tasks finish and it periodically sets up fetcher threads, one per node manager, which obtain the output relevant to the reducer from it's corresponding mapper. If the mapper output is smaller than a given amount, it resides in the memory. Otherwise, it is stored in the disk. These inputs are merged and the reduce function provided by the user is then called on these inputs. The output is then written to an HDFS file.

The entire mapreduce task involves a lot of shifting of data through the different nodes. It is imperative to identify and safeguard each of these flows in order to validate the entire procedure and guarantee confidentiality and integrity of data in the presence of a motivated attacker.

Chapter 3

Work Done

We have setup a two-node Hadoop cluster to run the experiments. In order to get a high level overview of how information flows in Hadoop, we have analyzed some log files generated by Hadoop. We have written a java program for performing WordCount in which we define the functionality of a Mapper and a Reducer. We also use this program to specify configuration details of the Hadoop cluster. We run this program on the two node Hadoop cluster to generate log files.

3.1 Rumen

For the purposes of log generation, we used a tool called Rumen. Rumen is a data extraction and analysis tool built for Hadoop. The Trace builder component of Rumen converts job history log files to easily parsed JSON format. Rumen extracts meaningful information from the JobHistory Logs created upon running a MapReduce job. A typical rumen output consists of details regarding every attempt of all the map and reduce tasks performed. For each map/reduce task, the log file contains information such as the number of bytes read, location of the node on which the task was performed, start and finish time of the task along with some other details.

3.2 Analyzing rumen output

We found the rumen output to be insufficient in its current state. A typical MapReduce program consists of several Map tasks followed by some Reduce tasks. The output(in the rumen log) corresponding to each task mentions the number of bytes read while performing this task. In Hadoop, the data is stored in the HDFS architecture in units of blocks. On the other hand, a split is a logical split of the data used during data processing by a Map

program. Every Mapper is assigned a particular split upon which the Mapper performs its operations. In cases where HDFS block size is less than the split size, a mapper might be assigned multiple HDFS blocks. In this case, the label of the mapper needs to be appropriately modified. To get an idea of how information flows, we think it is necessary to label each data split with the HDFS blocks it corresponds to. In the current rumen output, we can only see the number of bytes processed by a particular Mapper/Reducer and cannot see which HDFS blocks this corresponds to.

The label of a Mapper should be related to all the HDFS blocks it operates on. Each Mapper writes its output into an intermediate file which is stored on the local file system. These intermediate outputs are sent to the reducer for further operations. We need to label the intermediate outputs using the label of Mappers which generate this. The writers part of the label of intermediate data would then reflect the HDFS blocks of the data it corresponds to. Using these labels, we then can deduce the labels of reducers.

Chapter 4

Conclusion and Future Work

Given the above work and literature survey, we have finally come across the conclusion that the Rumen logs generated did not provide sufficient information to draw fine-grained data flows. We get an indication, but to provide a better solution, we need more informative logs.

Secondly, given the large and diverse variety of data flows that occur in a single map-reduce task, it will be necessary to safeguard each one of them. Since these flows may occur before the map phase and before and after the reduce phase, the logic to authorize them must be ingrained within Hadoop logic itself.

Possible leads which can be worked upon further are given as follows

- We feel that HDFS logs can provide us some insights since all data flows will eventually be captured as flows across HDFS data nodes. Perhaps the information flow control can also be employed within HDFS itself.
- A lot of intermediate data is also generated on Hadoop, both for the mapreduce task and for logging purposes. This data can also cause a breach of confidentiality if leaked and has to be identified and included within the RWFM framework.
- Hadoop source code will have to be inspected in order to understand where the data flow guard can be inserted.