Log And NetFlow Monitoring In Data Center Networks

A Dissertation Submitted in partial fulfillment of the degree of

Master of Technology
in
Computer Science

By
A.V Sriramya

School of Computer and Information Sciences
University of Hyderabad, Gachibowli
Hyderabad - 500046, India

June 28, 2013

CERTIFICATE

This is to certify that the dissertation entitled "**Log And NetFlow Monitoring In Data Center Networks**" submitted by **A. V Sriramya** bearing Reg. No. 11MCMT06, in partial fulfillment of the requirements for the award of Master of Technology in Computer Science, is a bona fide work carried out by him/her under my/our supervision and guidance.

The dissertation has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Dr. Anupama Potluri
School of Computer and Information Sciences,
University of Hyderabad

Dean,
School of Computer and Information Sciences,
University of Hyderabad

DECLARATION

I, **Sriramya** hereby declare that this dissertation entitled "**Log And NetFlow Monitoring In Data Center Networks**" submitted by me under the guidance and supervision of Dr. Anupama Potluri is a bona fide work. I also declare that it has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Date:

A. V Sriramya

Reg. No.: 11MCMT06

Signature of the Student

To,

My Parents and Supervisor.

Acknowledgments

I take this opportunity with immense pleasure to convey my gratitude to my supervisor, **Anupama Potluri**, who helped me throughout my project with her valuable suggestions. Her patience and constant guidance helped me to develop both academically and personally.

I would like to thank **Aravinda Potluri** (MD, INDIA Development Center, One Convergence) who gave the exposure to me regarding new technologies and helped me in understanding those technologies.

I am extremely grateful to our Dean, **Prof. Arun K Pujari**, for providing excellent computing facilities and such a nice atmosphere for doing my project.

Last but not the least, I would like to express my heartful wishes to my beloved parents and my dear friends, especially my lab mates, whose gracious solicitude helped me to complete this project successfully.

A.V Sriramya

Abstract

A data center consists of thousands of physical servers, each hosting tens of Virtual Machines (VMs). Each physical server has a software switch such as the Open vSwitch or the VMware vSwitch that connects the VMs to each other as well as to the outside world through the physical interface(s) of the server. These switches are typically OpenFlow enabled through the Floodlight controller. This helps in centralized management of the switching of packets within the data center. The data center leases these VMs to different tenants on request. It is essential that the information belonging to different tenants is kept separate to preserve confidentiality of tenants' data.

Each of the components in the data center – the VMs, software switches, Floodlight controller, physical interfaces, physical switches and routers – logs information such as any critical errors, errors, warnings etc. using *syslog*. Data center administrators who troubleshoot problems that arise in various components make use of these log messages to determine the causes. Unlike in typical corporations, there are two major challenges for log message collection, storage and monitoring in data center networks. Firstly, the amount of logging is extremely high due to the scale of operations in a data center. This requires scalable collection and storage. Secondly, the log information of the components of a tenant must be available to the tenant administrator. But, it is very important that the tenant administrator should not be able to access the logs of another tenant.

In this project, we use the free and open source log collection software *flume-ng* and extend it for use in data centers. To achieve this, we have added the capability to *flume-ng* to distinguish the tenant from the component IDs. This is done by querying the tenant ID from a Flow Isolation server designed by us that takes the component information as input and returns the tenant ID. The Flow Isolation Server is able to get the information that maps a component to a tenant ID from a configuration file that the data center, i.e., the cloud administrator provides. We use Cassandra as the storage backend for *flume-ng*. We made two major extensions to the data model

used by *flume-ng*: firstly, we extended it to store different tenant data in different Cassandra keyspaces. Secondly, we have columns that represent the component ID and severity in addition to the timestamp information for each event. This allows us to execute more fine-grained queries such as all logs of a particular severity or of a particular component etc.

The second part of this project is the development of a web based monitoring tool. The tool allows monitoring of log messages and NetFlow data. This tool uses LDAP for authentication. Based on whether a cloud adminstrator or a tenant administrator has logged in, the view shown is different. Whereas a cloud administrator can view all data of all clients, the tenant administrator is restricted to access data belonging to only the components belonging to his/her organization. We use JSP and JQuery to build this tool. CQL is used to query the Cassandra database and retrieve log messages and NetFlow records.

In future, this tool can be extended to raise alarms directly from *flume-ng* agents. Further, we can add NetFlow statistics to be computed on the fly and displayed rather than the raw NetFlow records. It can also be extended to display overloaded or underloaded physical server information based on the traffic pattern observed at a software switch. This will help the cloud administrator to load balance by either adding more VMs in other physical servers or consolidating VMs into fewer physical servers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A data center consists of thousands of Virtual Machines (VMs). These VMs belong to different tenants and are connected through software switches like Open vSwitch or VMWare vSwitch. Those switches are connected to software controllers like Floodlight. Logs generated by these components have to be collected and stored in databases for further analysis such that whenever en error occurs in the data center network a cloud administrator can troubleshoot the network. The amount of logging is extremely high due to the scale of operations in a data center. Thus, this requires scalable data collection and storage.

In a data center, it is essential that the information belonging to different tenants is kept separate to preserve confidentiality of tenants' data. But in typical corporations, they contain a single tenant and the scale of operations is very less as compared to data center networks. There is a tenant administrator to handle the typical corporations. In a data center, there will be a cloud administrator who has the control over all tenants' data and a tenant administrator who has the control only on his own tenant's data.

1.1 Motivation

The purpose of this project is to extend a log collection tool such that it works in data center networks and also to provide a log monitoring tool to monitor these collected log for troubleshooting the data center networks. The existing log collection tools are incapable of distinguishing the logs of various components as belonging to different tenants on the fly. It cannot identify the tenant ID of the component based on the component information which is available in the log message. But in data centers, it is very important that the tenant administrator should not be able to access the logs of another tenant. To provide privacy and security, we have to isolate the tenants logs while collecting and storing. To satisfy the above requirement, we have extended a log collection tool such that it is able to isolate tenants' logs while storing into a scalable database.

The monitoring tool is developed on top of Cassandra to monitor log and NetFl data. The NetFlow data is stored in the database through NetFlow collectors like *ntop* or net flow clients like Open Vswitch. NetFlow monitoring is useful for troubleshooting the network as well as to keep track of network traffic and bandwidth utilization. Thus, this is helpful in traffic analysis and bandwidth monitoring.

1.2 Contributions

In this project, we explained why log collection in data centers is not possible with the existing log collection tools. The existing log collection tool does not distinguish the components among the tenants in a data center. We use the free and open source log collection software *flume-ng* and extend it for use in data centers. To achieve this, we have added the capability to *flume-ng* to distinguish the tenant from the component IDs. To provide this capability we have developed a standalone server called Flow Isolation Server. Every *flume-ng* in the network communicates with this server to distinguish the log data of a component among the tenants. This is done by querying the tenant ID from a Flow Isolation server. It takes the component information as input and returns the tenant ID, keyspace etc.. The Flow Isolation Server is able to get this information from a configuration file that the data center, i.e., what the cloud administrator provides. We use Cassandra database as the storage backend for extended *flume-ng*. Cassandra is a highly scalable and available database with peer-to-peer architecture. We made two major extensions to the data model of Cassandra used by *flume-ng*. Firstly, we extended it to store different tenant data in different Cassandra keyspaces. Secondly, we have columns that represent the tenant ID, component , severity and timestamp in addition to the actual data information for each event.

The second part of this project is the development of a web based monitoring tool. The tool has two major applications which are log monitoring and NetFlow monitoring. This tool uses LDAP server for authentic a user. Based on whether a cloud adminstrator or a tenant administrator has logged in, the view shown is different. Whereas a cloud administrator can view all data of all tenants in data center networks, the tenant administrator is restricted to access data belonging to only the components belonging to his/her organization. The log monitoring provides three search options which are time period based search, severity based search and component based search. A user can perform a simple query by using any one of those search options or he/she can perform a complex queries by using combination of any search options.This allows a user to execute more fine-grained queries based on time period, severity and component. The NetFlow monitoring provides time period based search. A user can perform a query base on the time period to identify the NetFlow data within a particular time. For developing monitoring tool, we use JSP and JQuery. Cassandra Query Language(CQL) is used to query the Cassandra database and retrieve log messages and NetFlow records. CQl is SQL-like language and it supports SQL data definitions, data manipulations and SELECT syntax.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows: in Chapter 2, we present the abilities and limitations of *flume-ng* and advantages of NoSQL databases. We also present the logical and physical components in data center networks from which we collect log message information. In Chapter 3, we present the extensions to the log collection tool *flume-ng* and design and implementation of Flow Isolation server that allow *flume-ng* to distinguish between various tenants' information. In Chapter 4, we present the log and NetFlow monitoring tool. Finally, we conclude with Chapter 5.

Chapter 2

Related Tools

In this chapter, I describe a log collection tool and one of the NoSQL databases. I also describe the data center network components and connectivity between them. Syslog [9] message collection, storage and analysis has become essential in data center networks which involve virtual machines [12], Openv Switches [13] and Floodlight [5], the OpenFlow controllers. Storing and analyzing syslog messages help in identifying failures. $flume$[4] is one such tool for collecting, moving and storing syslog messages. This log data is stored in a scalable database for analysis. Cassandra [1] is one of the scalable and distributed databases to store these messages.

2.1 $flume$[4]

$flume$ is a reliable and distributed log collection service for efficiently moving large amounts of log data from its point of generation to a specified destination. The $flume$ agent is installed on every machine on which we want to gather the log data and the $flume$ collector is installed on a machine to collect the data from the $flume$ agents. There can be more than one $flume$ collector to collect log data from $flume$ agents. Every collector aggregates the data and sends it to a persistent data storage. All the nodes in the $flume$ cluster are controlled by a $flume$ master.

2.1.1 Flume Characteristics

$flume$ supports these four key features:

1. Reliability

2. Scalability

3. Manageability

4. Extensibility

2.1.1.1 Reliability

This property helps in continuing delivery of events in case of failures without losing data. *flume* accommodates fault-tolerance as its primary feature. It provides three reliability levels:

1. **End-to-end**

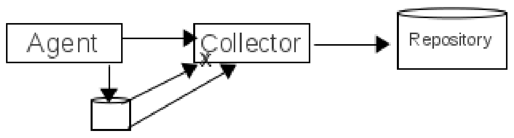


Figure 2.1: End-to-end reliability

The *end-to-end* reliability level guarantees that once *flume* accepts an event, it makes sure that event will reach the end point even in the event of multiple failures. The agent writes the events into a 'write-ahead log' and on the acknowledgment from the end point it deletes the events from it.

2. **Store-on-failure**

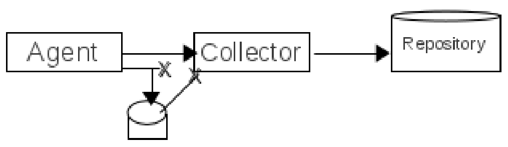


Figure 2.2: Store-on-failure reliability

The *store-on-failure* reliability level requires an acknowledgment from the next hop to delete data. If the sending node identifies any failure in the downstream node then it stores the events into the local disk until the node or path gets repaired.

3. **Best-effort**

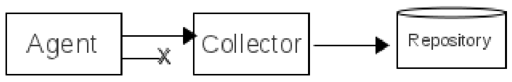


Figure 2.3: Best-effort reliability

The *best-effort* reliability level sends data to the next hop and removes the data. If any of the nodes fail, data that is in transit can be lost. It is the weakest reliability level but fastest because of lesser burden on the nodes.

2.1.1.2 Scalability

Scalability is the ability to increase system performance by adding more resources to the system. In a *flume* cluster, we can add any number of agents. If there is increased load for the collector, we can add more collectors.

2.1.1.3 Manageability

Manageability is the ability to control data flows. *flume* has centralized management point to monitor and change data flows between agents and the collector. *flume* Master is the point where the data flows can be managed. *flume* can dynamically reconfigure nodes by using the *flume* Master.

2.1.1.4 Extensibility

Extensibility is the ability to add new functionality to a system. *flume* provides many common input connectors(sources) and output connectors(sinks). It also provides flexibility to add new plug-ins as per the user requirement.

2.1.2 *flume* Node

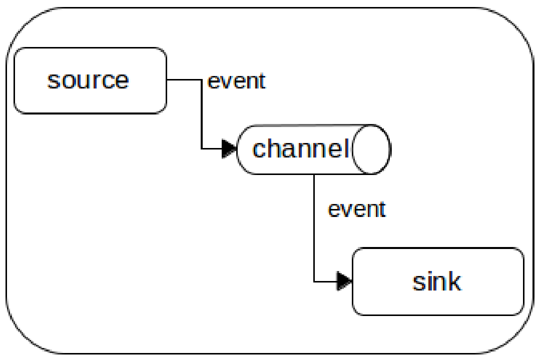


Figure 2.4: *flume* Node

Every logical *flume* node has three components: source, sink and channel. Source can be a file or a socket or a console. It represents the location from which *flume* gets the log message data and writes into the channel. Channel is a memory location or a file where it buffers the events. Then sink reads the events from the channel and sends it to the next hop. The next hop can be a *flume* collector or a *flume* agent or a repository.

flume internally converts every source of data into a stream of events. Events are *flume*'s unit of data. The event is composed of a body and header. The body of an event represents the actual content of the event and the header of the event contains meta data of the event.

2.1.3 Architecture

The architecture of *flume* is simple, distributed and flexible. *flume* uses stream-oriented data flow for transferring logs. Data flow starts from the *flume* agent and it passes through one or more *flume* collectors before it reaches its destination. Figure 2.5 shows the architecture of The *flume*.

The first tier in the architecture is the agent tier. Agent nodes are installed on machines that generate the logs. They forward data to the next tier, which is a tier of collector nodes. The collector nodes aggregate the separated data flows and forward them to the final storage tier.

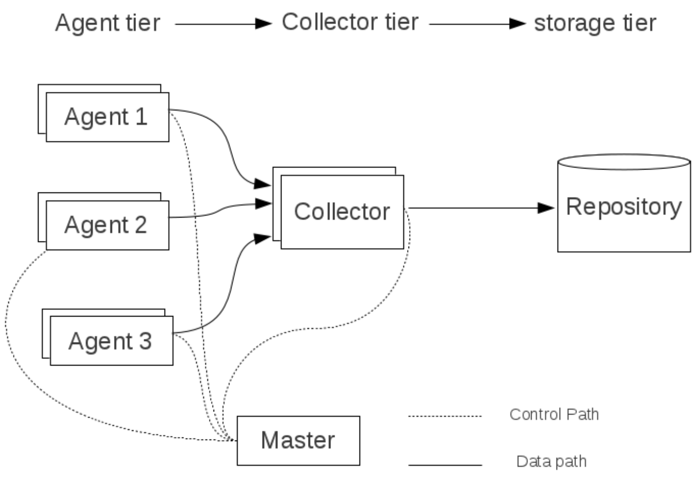


Figure 2.5: Architecture of *flume*

2.1.3.1 Master

It is a centralized configuration system for *flume* Agents and Collectors. It is a separate service with the knowledge of all the physical and logical nodes in a *flume* cluster. The Master assigns configurations to logical nodes. It provides an interface to the user for dynamically configuring the logical nodes. The logical nodes keep communicating with the master to share monitoring information.

2.1.3.2 Agent

Agent is a *flume* node. It can be either a logical or a physical node. It collects log data from machine and transfers to Collector. It has various types of sources and sinks. Agent source reads data and puts it into the channel while agent sink reads the data from channel and writes it in Collector's source.

2.1.3.3 Collector

flume collector collects log data from different Agent machines, aggregates data and transfers to a repository. Collector source gets the data from Agent and writes in to the channel. Collector sink reads the data from the channel and sends it to the repository.

2.1.3.4 Repository

Repository is a data base to store log data which arrives from various *flume* collectors. These repositories should have enough scalability to support the storage of log data which comes at high rates. NoSQL databases are preferable to store log messages which come from *flume* collectors.

2.1.4 Drawbacks

flume provides scalability while collecting logs through its agents. However, on the collector side, there is a bottleneck as the *flume* agents collect the data send to the collector at a high rate. In few cases collector is not capable of handling agents. It may lose events from those agents. Both the NoSQL databases used and the agents are scalable but the mediator which collects the data to store in the database is not scalable enough. To overcome this drawback, there is another version of *flume* available which is discussed in the next section.

2.2 *flume-ng* [3]

The modification is done to the architecture of the *flume* to provide high scalability. The *flume-ng* does not contain any centralized configuration system such as *flume* master and it does not contain the *flume* collector as well. It contains a *flume-ng* agent and it is capable of running any number of sources, sinks and channels between the source and the sink.

2.2.1 Architecture

The *flume-ng* node is same as old *flume*. In *flume-ng*, the source reads the data and it writes into the channel. Sink reads from the channel and writes into the database. *flume-ng* has various sources, sinks and channels. To store log messages from a

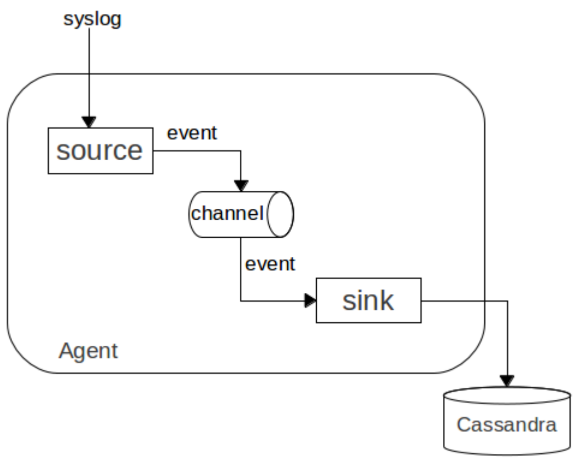


Figure 2.6: *flume-ng* architecture

machine to a database, we need to configure the source, sink and the channel. *flume-ng* agent has a configuration provider which provides the configuration parameters to configure these. To collect and store the syslog data into Cassandra using *flume-ng*, we use the following:

1. **Source:** Here, source is the *SyslogTcpSource*. It runs on a particular port number given in the configuration file. Whenever syslog daemon sends data to the *SyslogTcpSource*, the source reads that data, converts it into an event and writes into the channel. This source accepts two kinds of syslog message formats defined in RFC5424 and RFC3164 from the syslog daemon.

2. **Channel:** A Channel could a memory or a file. A channel in the form of a file is the most durable channel in case of power failures. Because this channel stores the events in secondary memory. The capacity of the channel is a configurable parameter. *flume-ng* gets the type of the channel and capacity of the channel from the configuration file. Whenever a channel exceeds the limit on the number of events that can be handled, the channel drops the events.

3. **Sink:** Here, sink is the *CassandraSink*. It reads the events from the channel and writes into Cassandra. It reads the configuration file to get the IP address

and port of the Cassandra database. It also gets the parameters needed to store events in Cassandra from the configuration file. Examples are Cassandra *Row key, Keyspace, Column family* etc.

4. **Event:** *flume-ng* event consists of an event header and an event body. Event body contains the actual data along with the component from which the data was generated. Event header consists of host IP address, severity value, facility value and unix time stamp.

5. **Configuration Provider:** This provides the configuration information to the *flume-ng* agents. In old $flume$, it is provided by the $flume$ master. Configuration provider removes the overhead of master. In a single configuration file we can give multiple sources, sinks and channels.

2.3 Cassandra [1]

Cassandra is a distributed NoSQL database system for storing and maintaining large amounts of data. It has been developed for solving the problem of Inbox search in Facebook. It combines Amazon Dynamo's with Google's Bigtable. It acquires the architectural properties from Amazon Dynamo and data model properties from Google's Bigtable. The data model of Cassandra is different from a traditional RDBMS data model.

Advantages of Cassandra:

1. High availability.

2. Massive scalability.

3. Dynamic Data model.

4. Data durability.

Cassandra provides the above described features because of its architectural properties and its data modeling properties.

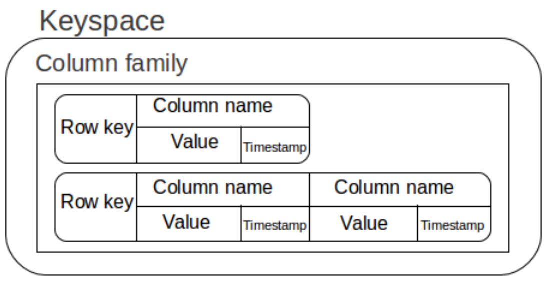


Figure 2.7: Cassandra Data model

2.3.1 Data Model

Cassandra stores the data in key-value pairs. It allows dynamic data modeling as per the data. It supports structured, semi-structured and unstructured data. It is case insensitive.

1. **KeySpace**: It is the container of column families. It is responsible for replicating the data in Cassandra. Keyspace should be unique in a Cassandra cluster.

2. **ColumnFamily**: It is defined in a keyspace and unique within a keyspace. It consists of the row keys and columns.

3. **Row key**: It is treated as a primary key in the Cassandra database. It is unique within a column family. Row key is associated with one or more columns based on the type of data. There is no restriction on the number of columns per row key within a column family. The choice of the node for the data storage is made on the basis of the row key.

4. **Column name**: Column name is associated with a column value and a Cassandra timestamp. This timestamp is used for maintaining consistency in the Cassandra database. Cassandra considers the higher timestamp valued data as the latest updated data.

2.3.2 Architecture

Cassandra is implemented as a cluster of nodes where the nodes are distributed across the world. Cassandra cluster has a distributed, peer-to-peer architecture and all nodes together form a ring shown in Figure 2.8. In this ring, all the nodes are symmetric and there is no centralized management control. Data is distributed and managed within a cluster by using data partitioning techniques.

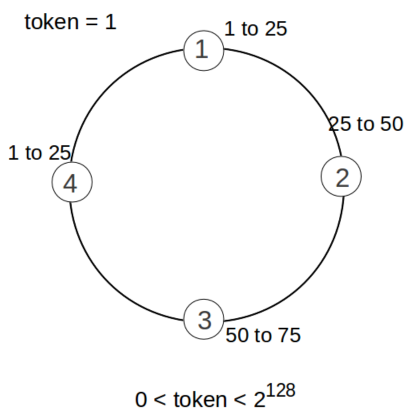


Figure 2.8: Architecture of Caasandra

2.3.3 Data Partitioning

Cassandra data partitioning consists of three elements.

1. **Topology**: Cassandra ring uses *gossip protocol* to discover the location and state information of the other nodes in a cluster. Using this protocol, nodes periodically exchange their state information and the other nodes get to know of their existence. This protocol helps in tracking the failure nodes in the cluster and rediscovers the node whenever it gets back to the cluster.

2. **Partitioner**: Data in the ring is divided by the number of nodes in the ring into ranges of key values. Each node is assigned with a token and each token is responsible for a range of data less than or equal to the token. Cassandra follows two types of partitioning techniques: *RandomPartitioner* and *ByteOrderedPartitioner*. In *RandomPartitioner*, the MD5 hash value of *row key* is compared with the token of every node. If the md5 hash value of *row key* is lesser than or equal to the token held by a node, it stores data in that node. In *ByteOrderedPartitioner*, byte order of the *row key* is compared with the token of every node. If the byte order value of *row key* is lesser than or equal to the token held by a node, it stores data in that node.

3. **Replica Placement strategy**: There are two types of replica strategy in Cassandra. First one is called *SimpleStrategy*. This strategy is used for single data center clusters. While creating *keyspace*, *SimpleStrategy* is mentioned in it along with the replication factor. Replication factor decides the number of replications in the Cassandra cluster. It places the first replica of data in a node which is determined by the partitioner. The remaining replicas are placed in subsequent nodes in the ring in clockwise direction. The second one is *NetworkTopologyStrategy*. This strategy is used for multi-data center cluster and it specifies the number of replicas in each data center. It places the first replica of data in a node which is determined by the partitioner. The remaining replicas are placed in subsequent nodes of the the data center and the nodes of other data centers in the ring in clockwise direction.

2.3.4 Operations on Cassandra Database

Cassandra performs read/write operations based on *row key*. Cassandra first stores data in a *commitlog* in the local disk and then transfers the data from the *commitlog* to *memtable* which is a main memory data structure. Memtable creates Sorted String Tables(SSTable) for each column family. It periodically writes data into SSTables and stores it in a sorted order of the *row keys*. The list of the operation performed on Cassandra is given below.

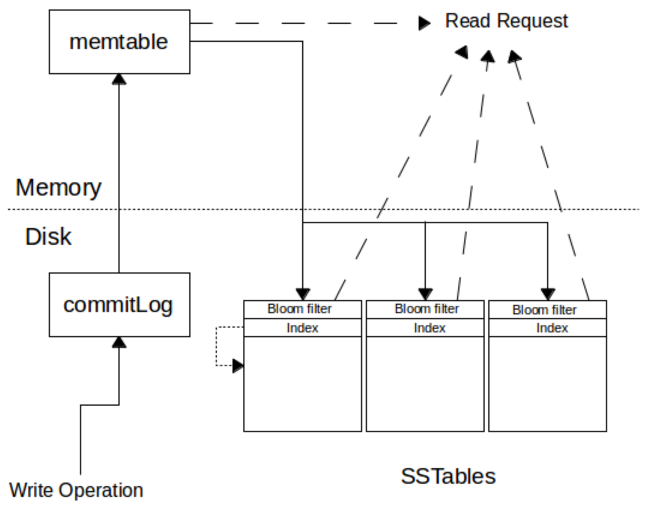


Figure 2.9: Read-Write Scenario in Cassandra

1. **Write Operation:** Write operation is successful only when Cassandra writes data into the commitlog and memtable. In the background it performs the other write operations like writing into the SSTable and maintaining the replicas. Cassandra provides tunable write consistency.

2. **Read Operation:** Cassandra performs read operations in two steps. First it checks if the requested data is available in memtable. If it is available it returns successfully; otherwise, it searches in SSTables. To increase the read performance, it uses the *bloom filter* for checking availability of the *row key* in SSTable and *index* to locate the data in SSTable. Cassandra provides tunable read consistency.

3. **Delete:** Whenever a delete request comes for Cassandra, it does not delete the data in the nodes. Instead, it marks the status of the data as *tombstone*. The data gets actually deleted while performing compaction.

4. **Compaction:** Cassandra performs the compaction operation to merge the SSTables within a node. This operation is performed by Cassandra internally Whenever the read performance of Cassandra goes down due to more number of SSTables in disk. During compaction it deletes the data which is marked as *tombstone* in SSTables.

2.4 Data center Network Components

A data center consists of thousands of physical servers, each hosting tens of Virtual Machines (VMs). Each physical server has a software switch such as the Open vSwitch or the VMware vSwitch that connects the VMs to each other as well as to the outside world through the physical interface(s) of the server. These switches are typically OpenFlow enabled through the Floodlight controller. This helps in managing the switching of packets within the data center. Each subsection in this chapter describes the functionality of the above software components.

2.4.1 Kernel Based Virtual machine(KVM) [12]

Kvm is a virtualization solution for linux and used for creating VMs in a physical server. It is a linux kernel module that turns the linux into hypervisor such that it supports virtualization support. It can scale well with extended hardware. It supports para-virtualization using virtio which increase its performance significantly. We have

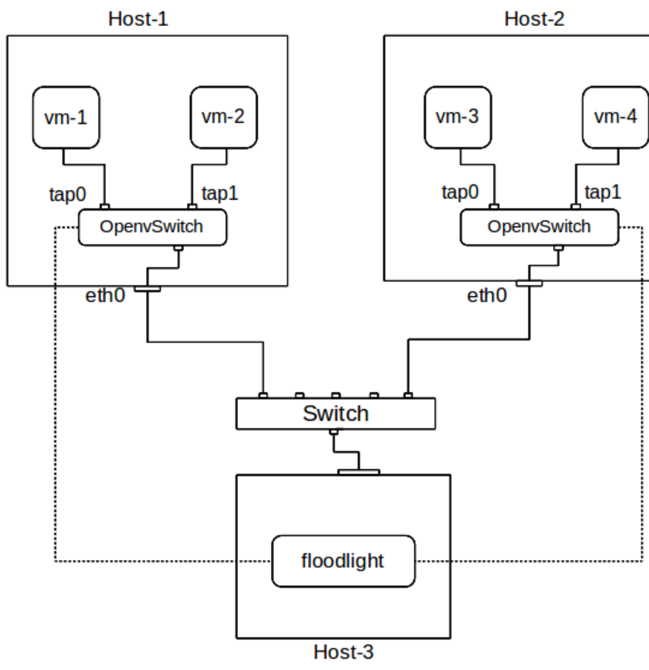


Figure 2.10: Virtual Network

used KVM for creating virtual network to simulate data center network environment with multiple tenants.

2.4.2 Openv Switch [13]

An Openv Switch is a multilayer software switch. It runs on a physical machine and acts as a switch in the data center environment. This virtual switch forwards the packets between virtual machines on same physical server and on different physical servers i.e. through physical network. With the support of Floodlight (defined in the followings subsection), it can differentiate between a data plane and a control plane. The control plane exists between the Floodlight and the Openv Switch and it is responsible for forwarding decisions. The data plane exists between the Openv Switch and the logical or physical machine and it is responsible for executing the commands which come from control plane.

2.4.3 Floodlight [5]

A Floodlight is a centralized software controller. It separates the data plane and control plane in a data center network. It presents a logical map of the entire network to control the applications implemented on the network. It supports both Openv Switch and Ethernet switch. Floodlight is mainly designed in order to support the growing number of physical switches, routers and software switches within the data center networks.

2.5 Summary

Tools which are used in the project,

1. We have used *flume-ng* in the project to collect log data from the data center network components.

2. Cassandra database has been used to store log data.

3. We have used software components like Open vSwitch, Floodlight, VMs to simulate data center networks for testing the software.

In the next chapter, we describe the extension of *flume-ng* to support separation of tenants' logs to different keyspaces in Cassandra. A new component called Flow Isolation Server is introduced to support this separation.

Chapter 3

Log Collection in Data Center Networks

A data center consists of thousands of Virtual Machines (VMs). These VMs belong to different tenants and are connected through software switches like Open vSwitch [13] or VMWare vSwitch [11]. Those switches are connected to software controllers like Floodlight [5]. Logs generated by these components have to be stored in Cassandra database for further analysis. We have chosen Cassandra database for storing log data due to its scalability, availability and data distribution. Whenever an error occurs in the network, the cloud administrator or a tenant administrator needs to monitor these logs to troubleshoot the networks. *flume-ng* is capable of collecting the log data in data center networks and storing the log data into Cassandra database. It is not suitable for log collection in data center networks due to the following reasons:

- It is incapable of differentiating the log data among the tenants when the data is collected from a data center network.
- It cannot identify the tenant ID of a component based on its IP address.
- Needs manual creation of keyspace for each tenant in Cassandra data base and defining the schema in it.
- Row key, keyspace and column family are needed to be specified manually in *flume-ng* configuration file. It might affect the security of the tenants.

To overcome the above limitations, we propose to have a server called Flow Isolation Server. Flow Isolation Server provides the support for isolation of information belonging to different tenants of the data center to different keyspaces. This helps in ensuring privacy to the tenants and does not compromise with security. It is described in Section 3.1. We have extended *flume-ng* by adding a Flow Isolation Client such that it can communicate with Flow Isolation Server to get information on which keyspace an event belongs to. We also modified its source and sink such that it supports the extended Cassandra data model which is described in Section 3.2. We have proposed this data model to provide a more convenient view of log data.

3.1 Flow Isolation Server

Flow Isolation Server (FIServer) is a standalone server which runs at a centralized location within a data center network. It maintains the information of all tenants and the related information like Cassandra keyspace and column families. Whenever a component generates a log event, this needs to be stored in the keyspace that is associated with the tenant that this component belongs to. A Flow Isolation client (FIClient), installed on every *flume-ng* agent, sends the component information to the FIServer and gets the Cassandra keyspace from it.

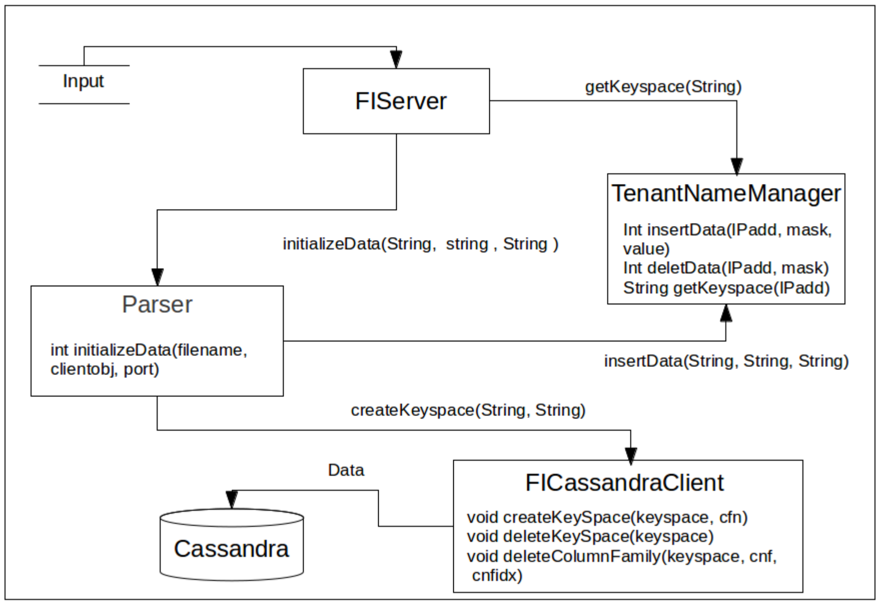

Input
FIServer
getKeyspace(String)
TenantNameManager
Int insertData(IPadd, mask, value)
Int deletData(IPadd, mask)
String getKeyspace(IPadd)
initializeData(String, string , String)
Parser
int initializeData(filename, clientobj, port)
insertData(String, String, String)
createKeyspace(String, String)
Data
FICassandraClient
void createKeySpace(keyspace, cfn)
void deleteKeySpace(keyspace)
void deleteColumnFamily(keyspace, cnf, cnfidx)
Cassandra

Figure 3.1: Architecture of the Flow Isolation Server

The architecture of Flow Isolation Server is shown in Figure 3.1. It consists of following modules.

1. FIServer Module: Communicates with the FIClient.
2. Parser Module: Reads the configuration file that maps different tenants' information to the corresponding Cassandra keyspace and column family.
3. TenantNameManager Module: Maps the given component information to the tenant ID and from that to the Cassandra keyspace and column family.
4. FICassandraClient Module: This is called by Parser module to create keyspace and column families in Cassandra for the tenant.

3.1.1 FIServer Module

The FIServer module is a daemon that initializes the flow isolation server with the help of the other modules. It then waits for requests from FIClient and serves them. It gets component IP addresses extracted from event headers by the FIClient associated with each *flume-ng* agent. It calls the *TenantNameManager* module to get the tenant ID and its related information like Cassandra keyspace and column family for the given IP address of the component. Then it sends the tenant ID and its related information to the FIClient of the *flume-ng* agent as a reply message.

3.1.2 Parser Module

This module is invoked by FIServer module. It gets the configuration file as input from the FIServer. The given configuration file contains the Cassandra server address, port number and FIServer port number. The configuration file also contains tenant IDs, the network IDs belonging to each tenant, the Cassandra keyspace and column family names of each tenant. The Parser module calls the *TenantNameManager* for each tenant specifying the network IDs that belong to it. It then calls *FICassandra-Client* to create the Cassandra keyspace and column families of this tenant.

initializeData (filename, client, port): This function is invoked by the FIServer module and takes configuration file as input. This API reads the configuration file, parses the data and stores the configuration information of the Cassandra server in a parameter called *client* and the port information of FIServer in a parameter called *port*. For each new network ID in the file, it calls an API from $TenantNameManager$ module to store the network ID and its related tenant ID, Cassandra keyspace and column family name. For every new tenant ID, it calls APIs from *FICassandraClient* module to create keyspace and column families in Cassandra.

3.1.3 TenantNameManager Module

This module manages the mapping between network IDs and their associated tenant IDs, keyspaces and column families. Each tenant can consist of many networks. The data center administrator can choose to store all the logs of a particular tenant in a single column family or multiple column families within a keyspace.

It provides the following APIs for the various functionalities supported:

1. **insertData** (networkID, mask, tenantID-keyspace-columnfamily): This function is invoked by the Parser module. It accepts the network ID, the subnet mask associated with this network ID and the Cassandra keyspace, column family and tenant ID that this network ID belongs to as input. These are then stored in the database for later use in retrieving the tenant ID, keyspace and column family information belonging to a given network ID.

2. **deleteData** (networkID): This function is invoked by Parser module and it accepts the network ID as input parameter. It deletes the given network ID from the database.

3. **getKeyspace** (ipaddress): This function is invoked by $FIServer$ module. It takes an IP address as the input parameter and searches for its tenant ID, Cassandra keyspace and column family.

3.1.4 FICassandraClient Module

This module acts as an interface between Cassandra and the Parser modules. It creates a connection with the Cassandra server. The data center administrator can create and delete keyspaces in Cassandra by the APIs provided by this module which are listed below.

1. **createKeySpace**(keyspace, columnfamily): This function is invoked by Parser module and it takes Cassandra keyspace and column family as input parameters. It creates the keyspace and the column family in Cassandra database.

2. **deleteKeySpace** (keyspace): This function is invoked by the Parser module and it takes keyspace as input parameter. It deletes existing keyspace in Cassandra database.

3. **deleteColumnfamily** (keyspace, columnfamily): This function is invoked by the Parser module and it takes column family and its keyspace as input parameters. It deletes the given column family from the keyspace in Cassandra database.

3.2 *flume-ng* Extension

We modified the *flume-ng* source such that it includes the application or component name of the log message in the *flume-ng* event header. We also modified the *flume-ng* sink such that it extracts elements like component IP address, component name, severity, facility etc. from the event header and it stores each element in a column within a column family of Cassandra database as per the proposed data model.

Proposed flume-Cassandra Data Model: In the proposed data model, each tenant has a keyspace. Log data which is related to a tenant goes to that keyspace. In a keyspace, a tenant can have a single column family or multiple column families to store the tenant logs. The column family which stores the log data is called *DataColumnFamily*. For every *DataColumnFamily*, Cassandra creates an index column family called *TimeindexColumnFamily* and it contains index for every event in *DataColumnFamily*. In *TimeIndexColumnFamily* each row key represents an hour. Row key of every event in *DataColumnFamily* is stored as a column in a row key of its *TimeIndexColumnFamily* based on its timestamp. It is an index that helps in performing time based queries while retrieving log data from Cassandra. Figure 3.2 depicts how the events are stored in Cassandra database.

3.2.1 SyslogTcpSource Modification

It is one of the *flume-ng* sources. It gets the port number from the configuration file of *flume-ng*. It opens a socket on the given port and waits for the log message which arrives from various components. Here, components are physical server, virtual machines, OpenvSwitch, floodlight etc.. The previous *SyslogTcpSource* does not extract the component or application name of the log message from the log before it creates a *flume-ng* event. The modified *SyslogTcpSource* extracts the component or application name from the log message and puts the component name into the event header while creating an event. Then it writes the event into the channel.

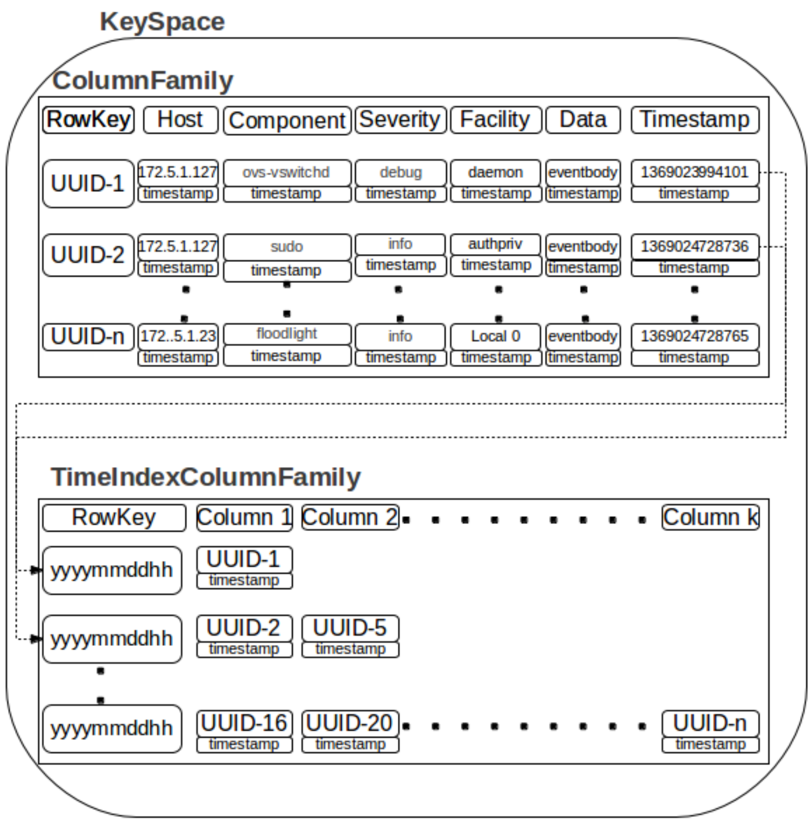


Figure 3.2: Flume-Cassandra Extended Data model

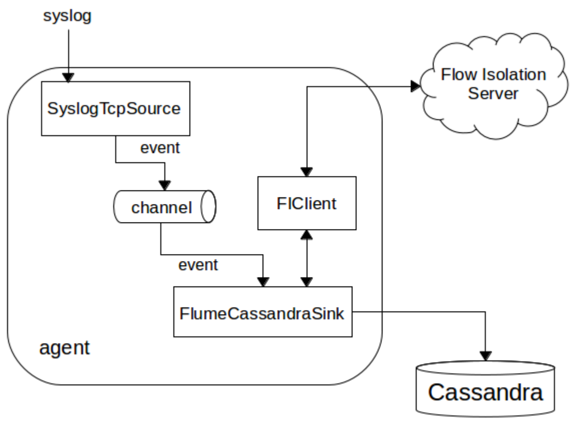


Figure 3.3: Architecture of *flume-ng* Extension

3.2.2 FlumeCassandraSink Modification

It is one of the *flume-ng* sinks. It reads data from the channel and stores it into Cassandra. The modification of *FlumeCassandraSink* reflects in the following modules.

1. **CassandraSink**: In the previous *flume-ng* agent, *CassandraSink* gets the Cassandra *Rowkey, keyspace, columnfamily* from the configuration provider and it stores the event in the given Cassandra *keyspace* and *columnfamily*. The *keyspace* and *columnfamily* need to be created in the Cassandra cluster manually before the agent starts storing the events. In the modified *CassandraSink* it does not get the *Rowkey, keyspace, columnfamily* from the configuration provider and it removes the burden of creating *keyspace* and *columnfamily* manually in Cassandra. It gets the Flow Isolation server address and port number from the configuration provider, calls *CassandraSinkRepository* module and passes these parameters to the module.

2. **CassandraSinkRepository**: In the previous *CassandraSinkRepository*, it gets the *Rowkey, keyspace, columnfamily* from the *CassandraSink* and stores the event body into the given Cassandra *keyspace* and *columnfamily* within a single *column*. It does not extract metadata from the event header. The modified *CassandraSinkRepository* gets the Flow Isolation server address and port number from the *CassandraSink*. Then it calls FIClient to get the Cassandra *keyspace* and the *columnfamily* based on the component IP address extracted from the *flume-ng* event header. Before storing an event into Cassandra, it extracts the component IP address, component, severity, facility and timestamp from the event header. It converts the severity and facility from their numerical values to actual names. Then the event body and the different elements of the event header are stored in different *columns* within a given *columnfamily* and *keyspace*. The *rowkey* for an event is the UUID[10] value of the event's timestamp.

3. **FIClient**: As part of the *flume-ng* extension, we added an FIClient to every agent. It gets the FIServer address and port number from the configuration provider. Then it creates a connection with FIServer. It sends the component IP address extracted from the *flume-ng* event header as a request message and it gets the tenant ID, Cassandra *keyspace* and *columnfamily* as a reply message.

In this chapter, we described the modification of *flume-ng* and design and implementation of the Flow Isolation Server. The modified *flume-ng* does log collection and storage in data center networks with the help of Flow Isolation Server and it stores the data in Cassandra with the proposed data model. In the next chapter, we describe the design and implementation of the monitoring tool which is built on top of Cassandra to monitor the stored logs.

Chapter 4

Log and NetFlow Monitoring in Data Center Networks

In the previous chapter, we discussed how the logs of components are collected and stored in Cassandra database. We also mentioned that, whenever an error occurs in a network, these collected logs are needed to troubleshoot the network. In this chapter, we describe a web based monitoring tool that allows monitoring of log messages and NetFlow records from Cassandra. NetFlow records are stored in Cassandra database by NetFlow collectors or NetFlow clients. We named the web based monitoring tool as CALONE-MON. It is an acronym for CAssandra LOg-data and NEtflow-record MONitoring. CALONE-MON allows a cloud administrator to view all data of all clients whereas, the tenant administrator is restricted to access data belonging to only the components belonging to his/her organization. CALONE-MON provides various search options to a user. A user can troubleshoot the network using log messages based on time period, severity and components. They can also use the NetFlow data to load balance the resources in the data center or detect anomalous network activity.

The chapter is organized as follows: In section 4.1, we describe the architecture of CALONE-MON and explain each of the components in it. We also explain the cloud administrator view and the tenant administrator view of CALONE-MON. Then in the next section we describe the two applications of CALONE-MON. In Section 4.3 we explain log monitoring and in Section 4.4 we explain NetFlow monitoring.

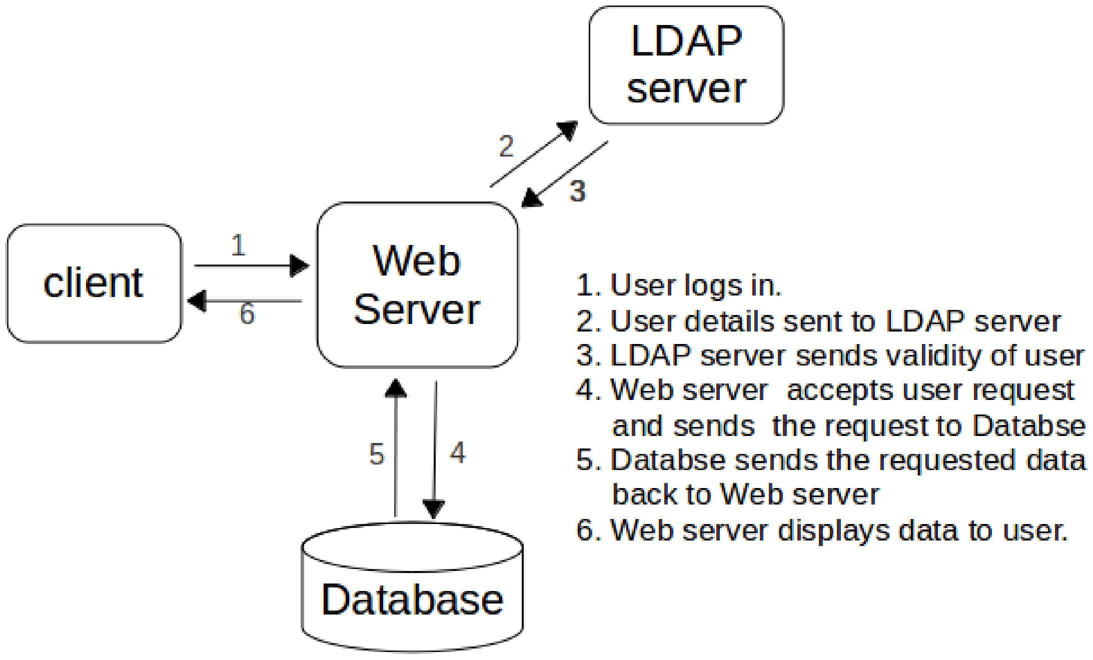

LDAP server
2
3
client
1
6
Web Server
5
4
Database
1. User logs in.
2. User details sent to LDAP server
3. LDAP server sends validity of user
4. Web server accepts user request and sends the request to Databse
5. Databse sends the requested data back to Web server
6. Web server displays data to user.

Figure 4.1: Architecture of the CALONE-MON

4.1 Architecture of CALONE-MON

The proposed CALONE-MON is installed on the web server to allow log and NetFlow monitoring of data center networks. It is a lightweight software, which consists of JSP and JQuery code. The installation and the maintenance of CALONE-MON is simple. The architecture of the CALONE-MON consists of the following components which are shown in Figure 4.1:

1. Lightweight Directory Access Protocol(LDAP) Server
2. Web Server
3. Client
4. Database

4.1.1 LDAP Server

We have used OpenLDAP 2.4.35 [**?**] software which is free and developed by OpenLDAP project. It is a standalone server to provide centralized authentication for a user to login from anywhere in the network. Its front end deals with network access and protocol processing and its back end deals with data storage. Whenever a user is registered in the LDAP server, it creates an entry in its own database. It stores all the users' information like name, password, uid, gid, givenName, homeDirectoryin etc. in a tree structure. The actual information of the user is stored in the leaf node of the tree structure. Whenever a request comes for a user directory information, in LDAP server, we need to specify the path between the top node and the leaf node of that user directory. LDAP receives this path information through its front end and passes it to the back end for processing. When the back end completes a request, it returns the result to front end. Then the front end sends the whole directory structure to the requested user as a reply.

4.1.2 Web Server

We have used Apache Tomcat as a web server and it is a servlet container developed by Apache Software Foundation(ASF). We are using Apache Tomcat version 7.0.41. Tomcat implements the Java Servlet and the Java Server Pages(JSP) specifications from Sun Microsystems and provides a pure JAVA and HTTP [6] web server environment to run Java code. It has support for packages like JQuery [7] . Web Server uses Cassandra Query Language(CQL) [2] to retrieve the data from data base.

4.1.3 Client

The client can be a cloud administrator or a tenant administrator. Whenever a cloud administrator logs in, he/she gets access to log messages and NetFlow records of all tenants. If a tenant administrator logs in, he/she gets access only for the corresponding tenant's log messages and NetFlow records.

4.1.4 Database

We have used Cassandra database. It is developed by Apache Software Foundation(ASF) and its version is 1.1.7. In a Cassandra cluster, the web server can communicate with any Cassandra node for accessing the data. Cassandra supports CQL and

CQl is SQL-like language and it supports SQL data definitions, data manipulations and SELECT syntax.

4.2 How CALONE-MON works

CALONE-MON supports two major applications which are *Log Monitoring* which is described in Section 4.3 and *NetFlow Monitoring* which described in Section 4.4. It does the following steps commonly for both the applications:

1. First, the login page is displayed whenever a user types the URL of the CALONE-MON in a browser. The login page is illustrated in Figure 4.2

2. Then the *username* and *password* are accepted by the CALONE-MON and these are sent to the LDAP server to authenticate the user.

3. Then LDAP server returns an exception to the web server if the user is not valid or if the given *password* is wrong. The web server, in turn, returns the corresponding error message to the user.

4. If the user is valid, the LDAP server returns the user information to the web server.

5. Based on whether a cloud administrator or a tenant administrator has logged in, the web server loads the corresponding JSP page. A cloud administrator has a choice to select a single tenant or all tenants to view the log and NetFlow data, but the tenant administrator gets access to his/her tenant log or NetFlow data.

6. Users have to choose either *Log Monitoring* or *NetFlow Monitoring* application to access the log or NetFlow data.

4.2.1 Cloud Administrator View

A user can choose either *All Tenants* view or a single tenant view (names of the tenants are displayed) for both *Log Monitoring* shown in Figure 4.3 and *NetFlow Monitoring* shown in Figure 4.4. If the user selects *All Tenants* view, the CALONE-MON allows the user to access the log or NetFlow data of components which belong to all the tenants in the data center network. If the user selects a particular tenant, the CALONE-MON allows the user to access the log or NetFlow data of components which belong to the selected tenant.

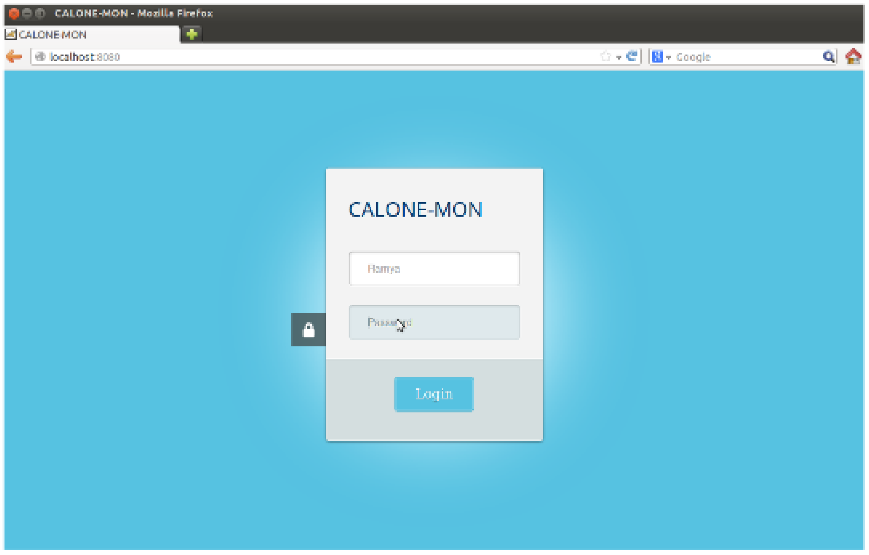


Figure 4.2: Login page

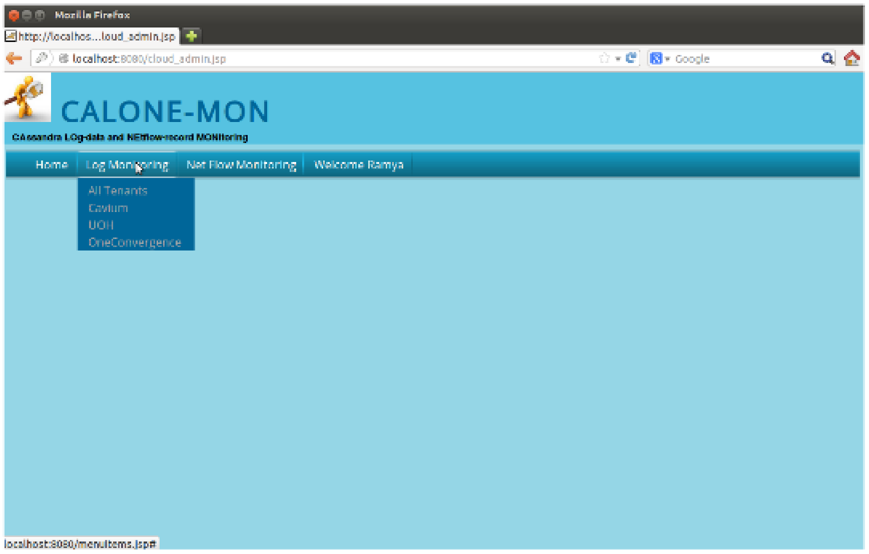


Figure 4.3: Cloud administrator view in Log Monitoring

4.2.2 Tenant Administrator View

A user is allowed to view the log or NetFlow data belonging to only the components belonging to his/her organization. User needs to select either *Log Monitoring* or

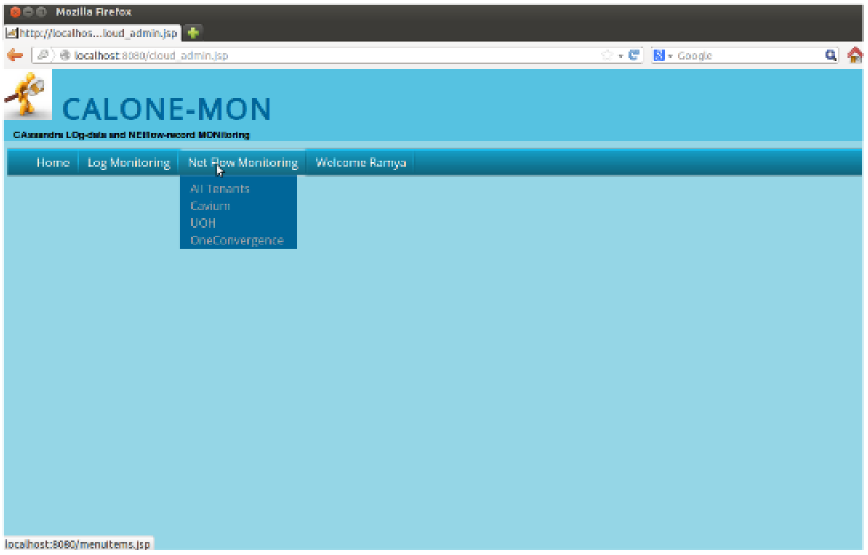

Mozilla Firefox
http://localhos...loud_admin.jsp
localhost:8080/cloud_admin.jsp
Google
CALONE-MON
CAssandra LOg-data and NEtflow-record MONitoring
Home
Log Monitoring
Net Flow Monitoring
Welcome Ramya
All Tenants
Cavium
UOH
OneConvergence
localhost:8080/menuItems.jsp

Figure 4.4: Cloud administrator view in NetFlow Monitoring

NetFlow Monitoring application to access the log or NetFlow data. which is shown in Figure 4.5.

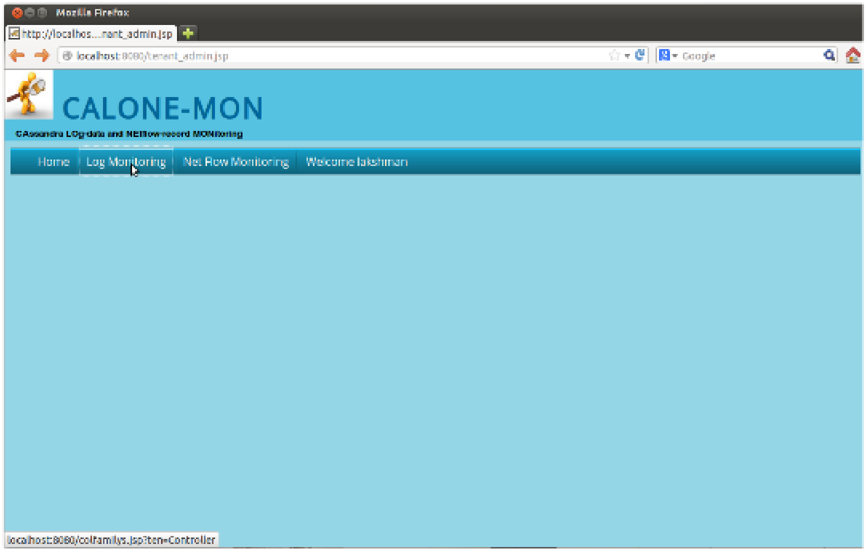
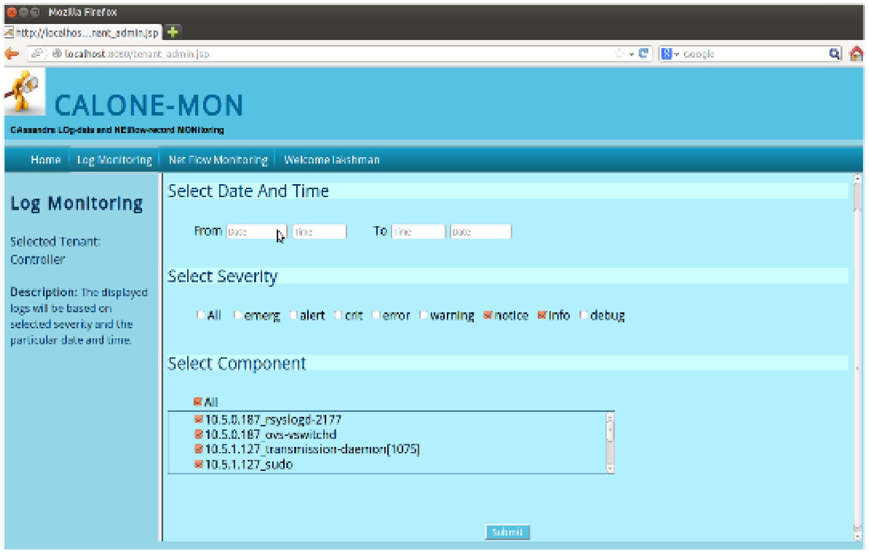

Mozilla Firefox
http://localhos....nant_admin.jsp
localhost:8080/tenant_admin.jsp
Google
CALONE-MON
CAssandra LOg-data and NEtflow-record MONitoring
Home
Log Monitoring
Net Flow Monitoring
Welcome lakshman
localhost:8080/colfamilys.jsp?ten=Controller

Figure 4.5: Tenant administrator view

4.3 Log Monitoring of CALONE-MON

Whenever a user selects *Log Monitoring* application, CALONE-MON displays a page which is shown in Figure 4.6. It provides three types of search options, which are give below:

- Search based on a *timeperiod*.
- Search based on *severity* of log message: user can choose one or more severity levels.
- Search based on *componentorapplication* of the log message: user can choose one or more components.



Log Monitoring

Selected Tenant: Controller

Description: The displayed logs will be based on selected severity and the particular date and time.

Select Severity

☐ All ☐ emerg ☐ alert ☐ crit ☐ error ☐ warning ☒ notice ☒ info ☐ debug

Select Component

☒ All
☒ 10.5.0.187_rsyslogd-2177
☒ 10.5.0.187_ovs-vswitchd
☒ 10.5.1.127_transmission-daemon[1075]
☒ 10.5.1.127_sudo

Figure 4.6: Search options in Log Monitoring

CALONE-MON allows the user to perform simple search as well as complex search. In simple search, the user can select any one of the search options. In complex search, the user can select combination of the search options. The displayed search results vary based on the type of search option selected, which are given below:

- Simple Search:

1. If user selects query based on time period, CALONE-MON fetches the event logs which belong to the selected time period from Cassandra database and displays it to the user.
2. If user selects query based on severity, CALONE-MON fetches the logs with the matching severity.
3. If user selects query based on component, CALONE-MON fetches the logs belonging only to that component name.

- Complex Search:

1. If user selects query based on time period and severity, CALONE-MON fetches the logs which have occurred during the selected time period. Before displaying logs to the user, it applies severity filter on the fetched logs to display the logs where severity matches with the selected severity.
2. If user selects query based on time period and component, CALONE-MON fetches the logs which have occurred during the selected time period. Before displaying logs to the user, it applies component filter on the fetched logs to display the logs where component is matched with the selected severity or component.
3. If user selects query based on time period and severity and component, CALONE-MON fetches the logs which have occurred during the selected time period. Before displaying logs to the user, it applies filters on the logs to display the log data where severity and component is matched with the selected severity and component.
4. If user selects query based on severity and component, CALONE-MON fetches the logs where the severities and the components of logs are matched with the selected severities and components and it displays to the user.

4.4 NetFlow Monitoring of CALONE-MON

Before describing about *NetFlow monitoring*, we describe NetFlow, the format of NetFlow records and how they are getting stored in Cassandra database. NetFlow is an IP flow monitoring protocol. Packets with the same source/destination IP address, source/destination ports, protocol interfaces and types of service are grouped into a flow. A flow in a NetFlow is also known as a *record*. These NetFlow records are stored in Cassandra by NetFlow collectors like *ntop* [?] or NetFlow clients like *OpenvSwitch* [13]. The NetFlow record format [8] is given below in Table 4.1

Table 4.1: NetFlow Record Format

Name	**Length in Bytes**
Source Address	4
Destination Address	4
Next Hop	4
Input	2
Output	2
Packet Count	4
Byte Count	4
First	4
Last	4
Source Port	2
Destination Port	2
Pad1	1
TCP Flags	1
Protocol	1
TOS	1
Source AS	4
Destination AS	4
Source Mask	4
Destination Mask	4
Pad2	2

4.4.1 NetFlow Cassandra Data model

In the proposed data model, each tenant has a keyspace. NetFlow data which is related to a tenant goes to that keyspace. In a keyspace, the tenant can have single column family or multiple column families to store the tenant NetFlow records. The column family which stores the NetFlow records is called *RecordColumnFamily*. For every *RecordColumnFamily*, Cassandra creates an index column family called *TimeIndexColumnFamily* and it contains index for every record in *RecordColumnFamily*. In *TimeIndexColumnFamily* each row key represents a hour. Row key of every record in *RecordColumnFamily* is stored as a column in the srow key of its *TimeIndexColumnFamily* based on its timestamp. It helps in performing time based queries while retrieving NetFlow data from Cassandra.

Figure 4.7 depicts how the NetFlow records are stored in Cassandra database.

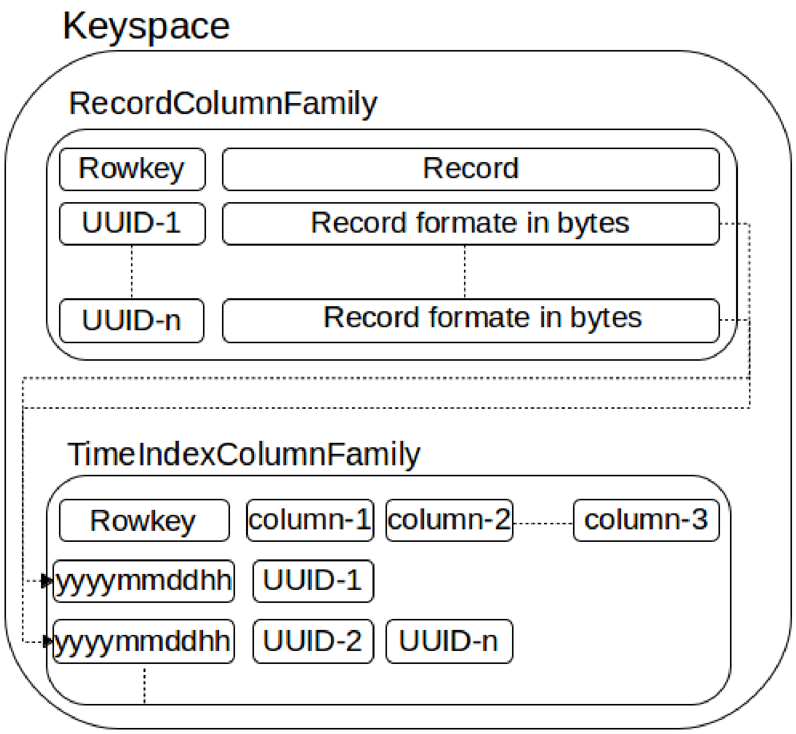
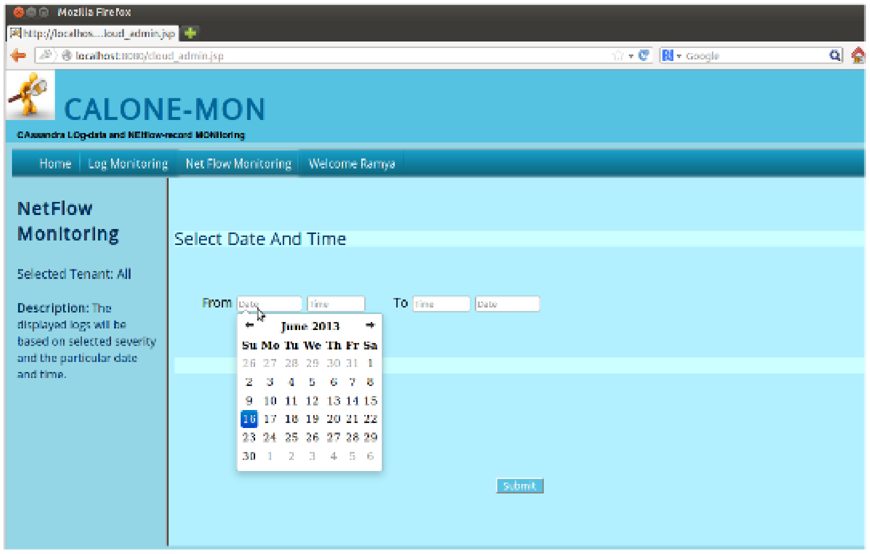


Figure 4.7: NetFlow Cassandra Data Model

4.4.2 NetFlow Monitoring

NetFlow monitoring is useful for troubleshooting the network as well as to keep track of network traffic and bandwidth utilization. Thus, it is helpful in traffic analysis and bandwidth monitoring. Whenever a user selects *NetFlow Monitoring* application, CALONE-MON displays a page which is shown in Figure 4.8. It provides a search option to the user based on time period. When the user selects a time period and requests for data, CALONE-MON fetches the NetFlow records which belong to the selected time period.

Summary: In this chapter, we described the design and implementation of a web application named CALONE-MON and explained its components. We also presented the cloud administrator and the tenant administrator view of log and NetFlow monitoring using CALONE-MON. The Log monitoring application offers search options based on Time period, severity and components. The NetFlow Monitoring applica-



Selected Tenant: All

Description: The displayed logs will be based on selected severity and the particular date and time.

Figure 4.8: Search option in NetFlow Monitoring

tion offers only time period based search option. It can be extended to provide load balancing and statistics of NetFlow record etc..

Chapter 5

Conclusion and Future work

5.1 Conclusion

Scalable log collection and storage is essential in data center networks. The logs which are generated by components of different tenants need to be stored in scalable database for further analysis. We have used a scalable and distributed database, Cassandra, to store the log data. Though the existing log collection tools are scalable, for collecting and storing log messages, they are not suitable for data center networks because of their inability to separate different tenants' data. We have introduced a standalone server named Flow Isolation Server and extended a log collection tool named *flume-ng* in order to overcome the above limitation and to store different tenants' data in different Cassandra keyspaces. The extended *flume-ng* gets the component's tenant ID by sending the component IP address to the Flow Isolation server. The Flow Isolation Server is able to get this information from a configuration file provided by the cloud administrator of the data center.

We have designed and implemented a monitoring tool named CALONE-MON on top of Cassandra database to monitor the log and NetFlow data. These NetFlow records are stored by the NetFlow collector or NetFlow client. CALONE-MON provides time based, component based and severity based search options for the log monitoring application. It provides time based search options for the NetFlow monitoring application. Whenever an error occurs in the network, these two monitoring applications are useful to a cloud administrator or a tenant administrator to troubleshoot the network.

5.2 Future Work

We can extend CALONE-MON such that it allows a cloud administrator to perform operation on Flow Isolation Server. List of possible operations are given below:

- Addition of a new tenant ID and deletion of an existing tenant ID.
- Creation and deletion of a keyspace in Cassandra.
- Addition and deletion of a component from a tenant.
- Creation and deletion of a column family in Cassandra keyspace.

We can also extend the Netflow monitoring application in CALONE-MON with the following ideas:

- Computation of NetFlow statistics on the fly and displaying them rather than the raw NetFlow records.
- Display of overloaded or underloaded physical server information based on traffic pattern observed at a software switch. This will help the cloud administrator to load balance by either adding more VMs in other physical servers or consolidating VMs into fewer physical servers.

Appendix A

Log collection tool extension

A.1 $flume - ng$ Extension

In *flume-ng* extension, we modified few modules in *flume-ng* and we also added a client to it. List of the modules are given given bellow.

1. SyslogTcpSource Module
2. CassandraSink Module
3. CassandraSinkRepository Module
4. FIClient Module

A.1.1 SyslogTcpSource Module

It is one of the *flume-ng* sources. It opens a socket on the given port and waits for *flume-ng* events. Whenever the source gets an event it writes the event into a channel. Here the channel is a file and its size is configurable. If the log data exceeds the channel capacity, it drops the events. The previous *SyslogTcpSource* do not extract the component from the event body before it writes to the channel. The extended *SyslogTcpSource* extracts the component from the event body before it writes into channel such that it provides the data in a particular format to the $flume - ng$ sink to store data in Cassandra which follows the proposed data model.

A.1.2 CassandraSink Module

This module gets the configuration parameters like Cassandra server address and port number from configuration file and stores it in a different variable. We have modified it in such a way that, it gets the Flow Isolation server address and port number along with the other parameters from the configuration file. Then, it calls CassandraSinkRepository module to pass the parameters.

A.1.3 CassandraSinkRepository Module

This module gets the configuration parameters from the CassandraSink module and $flume-ng$ event from the $flume-ng$ LogEvent module. Change in Cassandra data model reflects in this module. It extracts the various elements from the $flume-ng$ event header according to proposed data model. It calls FIClient module to get the Cassandra *keyspace* and the *column family* based on the IP address extracted from the $flume-ng$ event header. Then it calls CassandraClient to store data in Cassandra within the *keyspace* and the *column family.* While storing the data, it creates a *secondary index* on the *columns* of the *column family.*

- Interface Data Structure: None.
- Internal Data Structure: None.
- Interface functions: No modification done in interface functions.
- Internal functions:
    1. *void saveToCassandra(list eventList)* It takes the list of the events as input. For every event, it extracts the host IP address, component, severity, facility and time stamp from the event header. It converts the severity and facility from their numerical values to actual names. Then, it calls TimeUUIDUtils module to get the unique UUID for a time stamp as a Cassandra *Rowkey* and calls CassandraClient module to store the log data with the header information.
        (a) Input Parameters:
        eventList: List of the events in the channel.
        (b) Output Parameters: None.
        (c) Return Value: None.

2. *int initKeyspace(String host)* This function gets the IP address as input. Whenever a new $flume - ng$ event comes it calls FIClient to get the keyspace and column family. It creates *secondary index* on columns of every column family within a keyspace.
   (a) Input Parameters:
   host: Extracted host IP address from $flume - ng$ event.
   (b) Output Parameters: None.
   (c) Return Value: 0 on success, -1 on failure

A.1.4 FIClient Module

It makes a connection with the FIServer and get the keyspace and column family name by passing IP address of the host.

A.2 Flow Isolation

Flow Isolation is a part of *flume-ng* extension. It is a server, which runs in centralize location. As we discussed above, *flume-ng* does data transfer from its agent to Cassandra based on the *Row key, keyspace and column family* mentioned in its configuration file. By using FIServer(Flow Isolation Server), *flume-ng* gets its *keyspace* and $columnfamily$ automatically from the FIServer. Modules of Flow Isolation is given bellow.

1. FlowIsolation Module
2. Parser Module
3. TenantNameManager Module
4. FICassandraClient Module

A.2.1 FlowIsolation Module

This module takes configuration file as input and passes it to the Parser module. It starts a TCP Server and serves request which comes from *flume-ng agents*. It gets IP address as request message and sends the keyspace name resolved by the TenantNameManager module, as a reply message.

- Interface Data Structure None.

- Internal Data Structure None.
- Interface functions None.
- Internal functions
    1. *void FIServer (int port)* It takes port number as input and checks whether the port is available or not. If the given port number is available, it creates a socket with that port number.
        (a) Input Parameters:
        port: Given valid port number.
        (b) Output Parameters: None.
        (c) Return Value: None.
    2. *String search (String ip, TenantNameManager head)* This function takes IP address and reference to the TenantNameManager. It validates the IP address. On the success of validation, it calls APIs from TenantName-Manager module to get the respective tenant ID, keyspace and column family.
        (a) Input Parameters:
            - *ip:* Given IP address.
            - *head:* It is a reference to TenantNameManager.
        (b) Output Parameters: None.
        (c) Return Value: Keyspace-column family pair on success, null on failure.

A.2.2 Parser Module

This module invoked by FlowIsolation module. It takes configuration file as input and parses the data. While parsing data, it calls TenantNameManager module to initialize the data structure and on the success of that it calls FICassandraClient module to initialize the intial database scheema.

- Interface Data Structure: None.
- Internal Data Structure: None.
- Interface functions:
  int initializeData (String filename, TenantNameManager head, FICassandra-Client client, FlowIsolation port) It takes configuration file as input which comes

from the FlowIsolation module. It parses the data and stores the configuration information related to the Cassandra server, in client and the port information related to FIServer, in port. Input file contains network ID, tenant ID, a Cassandra keyspace and column family name. Whenever a new network ID appears, it calls APIs from TenantNameManager module.

1. Input Parameters:
   - *filename:* Configuration file. It consists of FIServer port number, Cassandra server address, port number and some other parameters like replication factor and strategy, which are related to Cassandra data storage. And it also consists of network IDs, associated tenant Id, keyspace and column family.
   - *head:* It is a reference to TenantNameManager.
   - *client:* It is a reference to FICassandraClient.
   - *port:* Given port number.
2. Output Parameters: None
3. Return Value: 0 on success, -1 on failure

- Internal functions: None.

A.2.3 TenantNameManager Module

This module manages mapping between network IDs and their associated tenant IDs, keyspaces and column family. Each tenant can consist of many networks. The data center administrator can choose to store all the logs of a particular tenant in a single column family or each network of a tenant in a separate column family with in a keyspace. Thus, in the latter case, there are many column families with in a keyspace for a single tenant.

- Interface Data Structure: None.

- Internal Data Structure: It uses binary tree data structure to store the tenant ID, keyspace and column family information.

- Interface functions:

1. *int insert (string networkID, int mask, string tenantID-keyspace-columnfamily)* This function accepts network ID, net mask and a Cassandra keyspace-columnfamily pair along with tenant ID as input. It maps the network ID to the given keyspace-columnFamily. All network IDs of the same tenant can be stored in the same or a different column family.
   (a) Input Parameters:
      - *networkID:* Given Network ID.
      - *mask:* Given Network mask.
      - *tenantID-keyspace-columnfamily:* Given tenant ID, keyspace and column family.

   (b) Output Parameters: None
   (c) Return Value: 0 on success, -1 on failure
2. *int delete (string networkID)* This function accepts the network ID. It deletes that particular network ID from the data structure.
   (a) Input Parameters:
      networkID: Given Network ID
   (b) Output Parameters: None
   (c) Return Value: 0 on success, -1 on failure
3. *string find (string ipaddress)* This function takes IP address of an event and searches for its network ID, tenant ID and keyspace-columnfamily pair.
   (a) Input Parameters:
      ipaddress: IP address of an event.
   (b) Output Parameters: None
   (c) Return Value: keyspace-columnfamily on success, -1 on failure

- Internal functions: None.

A.2.4 FICassandraClien Module

This module acts as an interface between the Cassandra and the Parser module. The data center administrator can create and delete keyspaces in cassnadra by the APIs provided by this module.

- Interface Data Structure: None.

- Internal Data Structure: None.
- Interface functions:
  1. *int createKeySpace (string keyspace, String columnfamily)* This function creates keyspace and column family in cassndra.
     (a) Input Parameters:
       - *keyspace:* Given keyspace name for a tenant.
       - *columnfamily:* Given column family name for a networkID of the tenant.

     (b) Output Parameters: None

     (c) Return Value: 0 on success, -1 on failure
  2. *int deleteKeySpace (string keyspace)* This function deletes keyspace in cassandra.
     (a) Input Parameters:
     keyspace: Existed keyspace for a tenant.

     (b) Output Parameters: None

     (c) Return Value: 0 on success, -1 on failure
  3. *int deleteColumnfamily (string keyspace, String columnfamily)* This function deletes column family within the keyspace in cassandra.
     (a) Input Parameters:
       - *keyspace:* Existed keyspace for a tenant.
       - *columnfamily:* Existed column family for a tenant.

     (b) Output Parameters: None

     (c) Return Value: 0 on success, -1 on failure
- Internal functions: None.

Appendix B

CALONE-MON

In CALONE-MON, we have used Cassandra Query Language for retrieving data from Cassandra database. We have used various CQL queries to perform various search operations in log and NetFlow monitoring. CQL is a SQL like language but it does not support the all kind of queries which supported by SQL. It is a subset of SQL.

B.1 Database connection

Before performing the queries CQL need a connection with Cassandra database. It creates the connection with Cassandra by using *getOrCreateCluster(ClusterName, CassPort).* Here, ClusterName is Cassandra server name and CassPort is port of the Cassandra server. Then CQL makes a connection with the keyspace given by CALONE-MON with the help of *createKeyspace(ksp, CassAdd).* Here, ksp is the given keyspace and CassAdd is the address of the Cassandra. For example, look at the bellow queries.

```
Cluster CassAdd = getOrCreateCluster(ClusterName, CassPort);

CqlQuery<String, String, String> cqlQuery = new CqlQuery<String,
     String, String>(createKeyspace(ksp, CassAdd), se, se, se);
```

B.2 CQL Queries

CALONE-MON performs three verities of CQL queries. Which are

1. Time period based search.

2. Severity based search.
3. Component based search.

Time period based search: This types of CQL queries does the search with the help of Index column families. CALONE-MON gets the start time and end time from the user then it extracts the hours within the start and end time. It send these hours as a row key to the CQL. CQL query need to get the column names of the given row key in a given index column family. As we explained in previous chapters, these column names are the row key of the data column family of the index column family. Based on the row key, it fetches the log data or NetFlow data from the data column families. Bellow we have explained how these queries are get executed:

```
multigetSliceQuery.setColumnFamily(<IndexColumnFamily>);

multigetSliceQuery.setKeys(<hoursList>);

QueryResult<Rows<String, UUID, String>> result =
                          multigetSliceQuery.execute();

Rows<String, UUID, String> orderedRows = result.get();

for (Row<String, UUID, String> r : orderedRows) {
 for(k=0; k<r.getColumnSlice().getColumns().size(); k++) {
  Key = r.getColumnSlice().getColumns().get(k).getName().toString();
  if (Key >= startTime && Key <= endTime)
  cqlQuery.setQuery("select * from <DataColumnFamily> where key = <Key>");
  QueryResult<CqlRows<String, String, String>> data = cqlQuery
                                                       .execute();
   if (data != null && data.get() != null) {
      java.util.List<me.prettyprint.hector.api.beans.Row<String, String,
                                 String>> list = data.get().getList();
      for (me.prettyprint.hector.api.beans.Row<String, String, String>
                                                        row : list) {
        for (i=1; i<=row.getColumnSlice().getColumns().size(); i++)
         display(row.getColumnSlice().getColumns().get(i).getValue());
      }
```

```
    }
  }
 }
```

In time period based search, if user selects only time period based search then the above queries are sufficient to performs that. But if the user selects severity or component or both along with the time period based search then CALONE-MON applies the filter on data fetched by CQL. Because CQL is not capable of performing *join* operations like SQL. For severity, CALONE-MON takes the selected severities and compares the severity of the data with the selected severities. If match is true it displays the data other wise it ignores the data. For component, CALONE-MON takes the selected components and compares the component of the data with the selected components. If match is true it displays the data other wise it ignores the data. For both severity and component, CALONE-MON takes the selected severities and components and compares the severity of the data with the selected severities and components. If match for both are true it displays the data other wise it ignores the data.

Severity based search: CALONE-MON takes the selected severities. It fetches the data for all selected severities and displays the data.

```
for(i=0; i < sevList; i++) {
 cqlQuery.setQuery("select * from <DataColumnFamily> where
                                     severity = sevList[i]");
 QueryResult<CqlRows<String, String, String>> data = cqlQuery
                                                   .execute();
 if (data != null && data.get() != null) {
     java.util.List<me.prettyprint.hector.api.beans.Row<String, String,
                               String>> list = data.get().getList();
     for (me.prettyprint.hector.api.beans.Row<String, String, String>
                                                    row : list) {
       for (i=1; i<=row.getColumnSlice().getColumns().size(); i++)
        display(row.getColumnSlice().getColumns().get(i).getValue());
     }
  }
```

```
   }
```

Component based search: CALONE-MON takes the selected components into a array. It fetches the data for all selected components and displays the data.

```
  for(i=0; i < compList; i++) {
   cqlQuery.setQuery("select * from <DataColumnFamily> where
                                     component = compList[i]");
   QueryResult<CqlRows<String, String, String>> data = cqlQuery
                                                       .execute();
   if (data != null && data.get() != null) {
       java.util.List<me.prettyprint.hector.api.beans.Row<String, String,
                                 String>> list = data.get().getList();
       for (me.prettyprint.hector.api.beans.Row<String, String, String>
                                                        row : list) {
         for (i=1; i<=row.getColumnSlice().getColumns().size(); i++)
          display(row.getColumnSlice().getColumns().get(i).getValue());
       }
    }
   }
```

Severity and Component based search: CALONE-MON takes the selected severities and components into two different array . It fetches the data for all selected severities and components and displays the data.

```
for(j=0; j < sevList; j++) {
  for(i=0; i < compList; i++) {
   cqlQuery.setQuery("select * from <DataColumnFamily> where
                 severity = sevList[j] and component = compList[i]");
   QueryResult<CqlRows<String, String, String>> data = cqlQuery
                                                       .execute();
   if (data != null && data.get() != null) {
       java.util.List<me.prettyprint.hector.api.beans.Row<String, String,
                                 String>> list = data.get().getList();
       for (me.prettyprint.hector.api.beans.Row<String, String, String>
                                                        row : list) {
         for (i=1; i<=row.getColumnSlice().getColumns().size(); i++)
         display(row.getColumnSlice().getColumns().get(i).getValue());
      }
   }
  }
}
```

Bibliography

[1] **Apache Cassandra Home page**. http://www.datastax.com/docs.

[2] **Cassandra Query Language**. http://cassandra.apache.org/doc/cql/CQL.html.

[3] **Flume 1.3.0 User Guide (Flume-ng)**. http://flume.apache.org/FlumeUserGuide.html.

[4] **Flume User Guide**. http://archive.cloudera.com/cdh/3/flume/UserGuide/#_the_flume_node.

[5] **Getting Started Floodlight**. http://www.projectfloodlight.org/getting-started/.

[6] **HTML**. http://www.w3schools.com/html/.

[7] **JQuery**. http://api.jquery.com/.

[8] **NetFlow Export Datagram Format**. http://www.cisco.com/en/US/docs/net_mgmt/netflow_collection_engine/3.6/user/guide/format.html.

[9] **Syslog**. http://en.wikipedia.org/wiki/Syslog.

[10] **UUID**. http://en.wikipedia.org/wiki/Universally_unique_identifier.

[11] **VMWare Vswitch**. http://www.vmware.com/in/products/datacenter-virtualization/vsphere/distributed-switch.html.

[12] Scott Lowe. **Installing KVM and OpenvSwitch on Ubuntu**. http://blog.scottlowe.org/2012/08/17/installing-kvm-and-open-vswitch-on-ubuntu/, AUG 17, 2012.

[13] Brent Salisbury. **Getting Started OpenFlow OpenvSwitch Tutorial**. http://networkstatic.net/openflow-openvswitch-lab/, JUN 15, 2012.