Data Efficient NSX Inventory State Management in Cross-Cloud Environment

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

In

Computer Science and Engineering

Submitted By

Navjot Kaur

(801532032)

Under the supervision of:

Dr. Ashima Singh

Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

July 2017

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "***Data Efficient NSX Inventory State Management in Cross-Cloud Environment***", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of ***Dr. Ashima Singh*** and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for the award of any other degree of this or any other University.

(Navjot Kaur)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

16/07/17

(Dr. Ashima Singh)

Assistant Professor

CSED, Thapar University

Patiala

ABSTRACT

In a typical NSX deployment, software assets such as virtual machines and virtual network interfaces of a distributed set of ESXi / KVM hosts need to be discovered, collated and persisted.

In traditional on premise deployments, as the scale of these host elements grows, it is impractical to persist all the attributes of the interested asset set on the NSX manager. It is more efficient to hold some of the attributes in memory and fetch them again in case the NSX manager crashes and recovers. The NSX manager itself is deployed in a high availability configuration with shared storage, thereby reducing the risk of loss of data. This product plans to support scenarios where these assumptions of re-populating the entire data set may not be cost effective in future. Take for instance a cross cloud / hybrid cloud deployment scenario, where the NSX Manager is deployed on the premises of an enterprise customer, which is managing SDN use cases for both on premise and public cloud workloads. In such a situation, inventory data must be collected from the public cloud environment and sent over WAN links to the on premise NSX manager. The product must evolve, to minimize the need for re-populating host inventory information as it encounters various faults.

The aim is to study the full sync scenarios and find the current size of inventory information for a typical NSX deployment and propose a solution to effectively reduce the inventory delta exchange.

ACKNOWLEDGEMENT

First of all I would like to thank the Almighty, who has always guided me to work on the right path in life.

This work would not have been possible without the encouragement and able guidance of my supervisor **Dr. Ashima Singh**. I thank my supervisor for their time, patience, discussions and valuable comments. Their enthusiasm and optimism made this experience both rewarding and enjoyable.

I am also heartily thankful to **Dr. Maninder Singh**, Associate Professor and Head, Computer Science & Engineering Department and **Dr. Ashutosh Mishra**, PG coordinator, for motivation and providing uncanny guidance and support throughout the preparation of the thesis report.

I will be failing in my duty if I don't express my gratitude to **Dr. S.S. Bhatia,** Senior Professor and Dean of Academic Affairs, Thapar University, for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field.

I am also thankful to the entire faculty and staff members of Computer Science and Engineering Department for their direct-indirect help, cooperation, love and affection, which made my stay at Thapar University memorable.

Last but not least, I would like to thank my family whom I dearly miss and without whose blessings none of this would have been possible. To my parents, I own thanks for their wonderful love and encouragement. I would also like to thank my brother, since he insisted that I should do so. I would also like to thank my close friends for their constant support.

Navjot Kaur

(801532032)

TABLE OF CONTENTS

Table of Contents **Page No.**

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

VM	Virtual Machine
VIF	Virtual Interface
vCPU	Virtual Central Processing Unit
ESX	Elastic Sky X
PXE	Preboot Execution Environment
VPN	Virtual Private Network
VC	Virtual Center
MP	Management Plane
MPA	Management Plane Agent
KVM	Kernel-based Virtual Machine
DA	Discovery agent
DC	Data Collector
DS	Data Sender

CHAPTER 1

INTRODUCTION

This chapter describes the virtualization and basic VMware products, which form an important part of virtualization. Also it describes inventory services provided in the cross-cloud environment. Last but not the least it describes the structure of thesis report.

1.1 Background

Virtualization means abstraction of resources. Virtualization techniques can be applied to networks, storage and compute. It is quite different from simulation and emulation. Simulation is creating a replica of a system. But it will not provide all the functionality that is provided by real system. It is mostly used for testing purpose. Emulation is representing entire hardware in software form. All the applications that run on hardware can now run on any machine having its software.

Compute Virtualization is creating a virtual environment for any program to run on an existing platform as a guest, without interfering or interrupting with the host platform's services or programs. It helps to reduce the cost because less hardware resources are required. Also it adds flexibility to the system. New virtual servers can be added in very less time. So it makes provisioning of resources and applications easier and faster.

The data center has vSphere Suite that includes vCenter Server, vSphere Client, NSX Manager and other NSX components.

1.1.1 VSphere

vSphere suite is composed of two main components namely vCenter Server and ESXi host. ESXi is a virtualization platform where virtual machines and virtual appliance can be created and run. VCenter Server acts as administrator of ESXi hosts. It manages resources of multiple hosts. With the help of vSphere vMotion virtual machines can be migrated to other host to avoid downtime.

Figure 1.1 shows the vSphere structure. vSphere client is connected to vCenter server. The vCenter Server consists of data centers which can have multiple hosts. Each host contains multiple virtual machines.

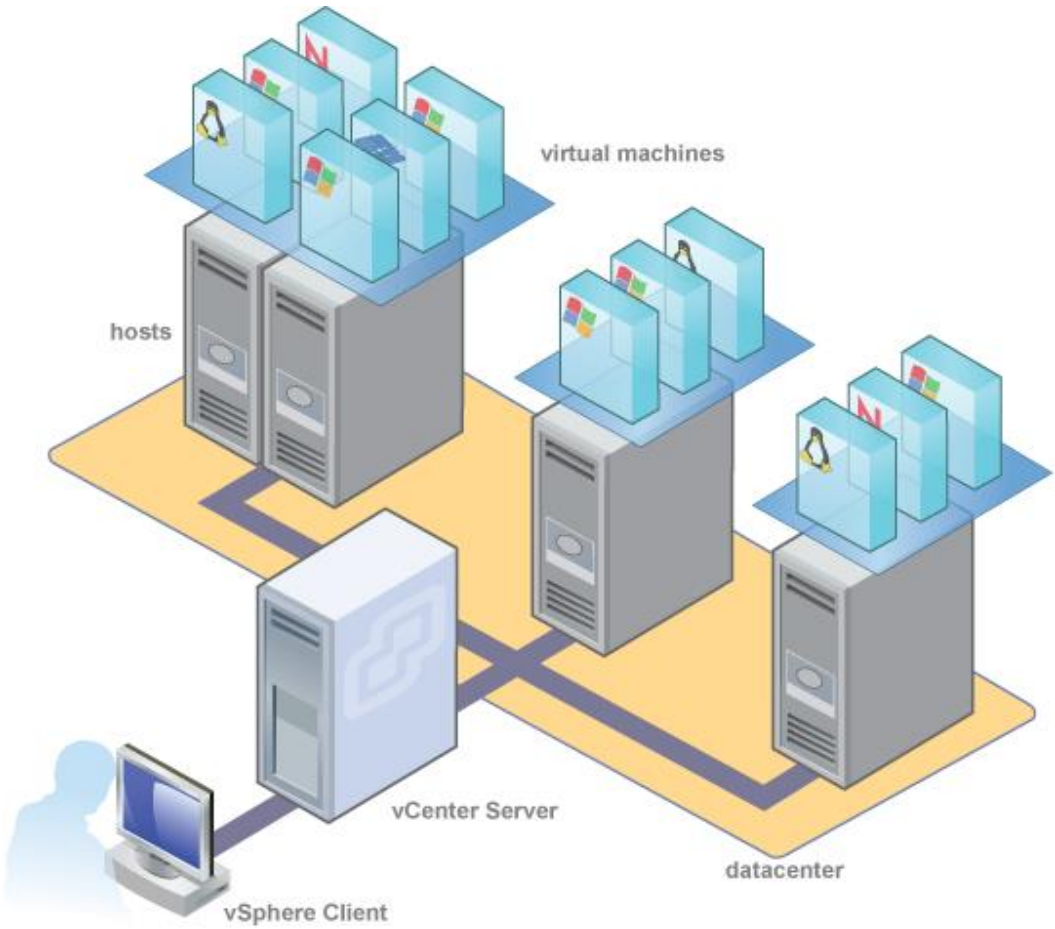

Figure 1.1 vSphere [1]

1.1.2 ESXi

ESXi is a bare metal hypervisor, which means that they run directly on server hardware and do not require the installation of an additional underlying operating system. ESXi server is present as a part of vSphere suite. The virtual machines are hosted on this server. The operating system is installed on this ESXi host and the applications are run.

Figure 1.2 shows the ESXi Server structure.

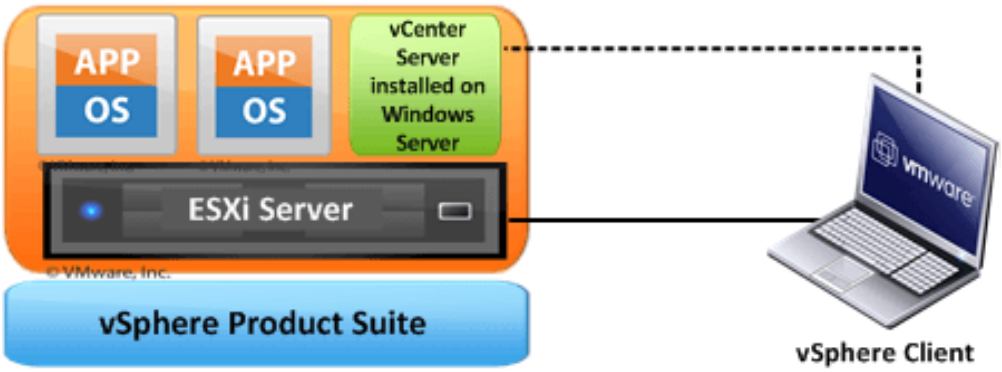

Figure 1.2 ESXi Server [1]

It is a virtualization platform which helps to run more servers on the same physical hardware leading to efficient utilization of the hardware resources such as CPU, memory etc. It allows running virtual machine by providing resources as and when required.

Physical machine's software representation called as virtual machine is the place where applications and operating system can run.

Infrastructure is what connects resources to the business. Virtual Infrastructure is a dynamic mapping of the resources to business. It results in decreased costs and increased efficiencies and responsiveness.

Various methods that can be used to deploy ESXi are interactive installation, scripted installation and network based installation that uses PXE.

1.1.3 vCenter Server

vCenter Server is required to manage vSphere environment. vCenter Server is a platform that manages virtual machines and hosts. It is responsible for management, operation, and provisioning resources to virtual machines and hosts. It can be installed on Microsoft Windows virtual machine or physical server.

Figure 1.3 shows the vCenter Server structure. vCenter server has a datacenter with hosts present on it. The virtual machines are then added to these hosts. After this the operating system is installed and the applications are run. Every NSX manager has one to one mapping with the vCenter. Both NSX manager and VC manage the hosts which run virtual machines.

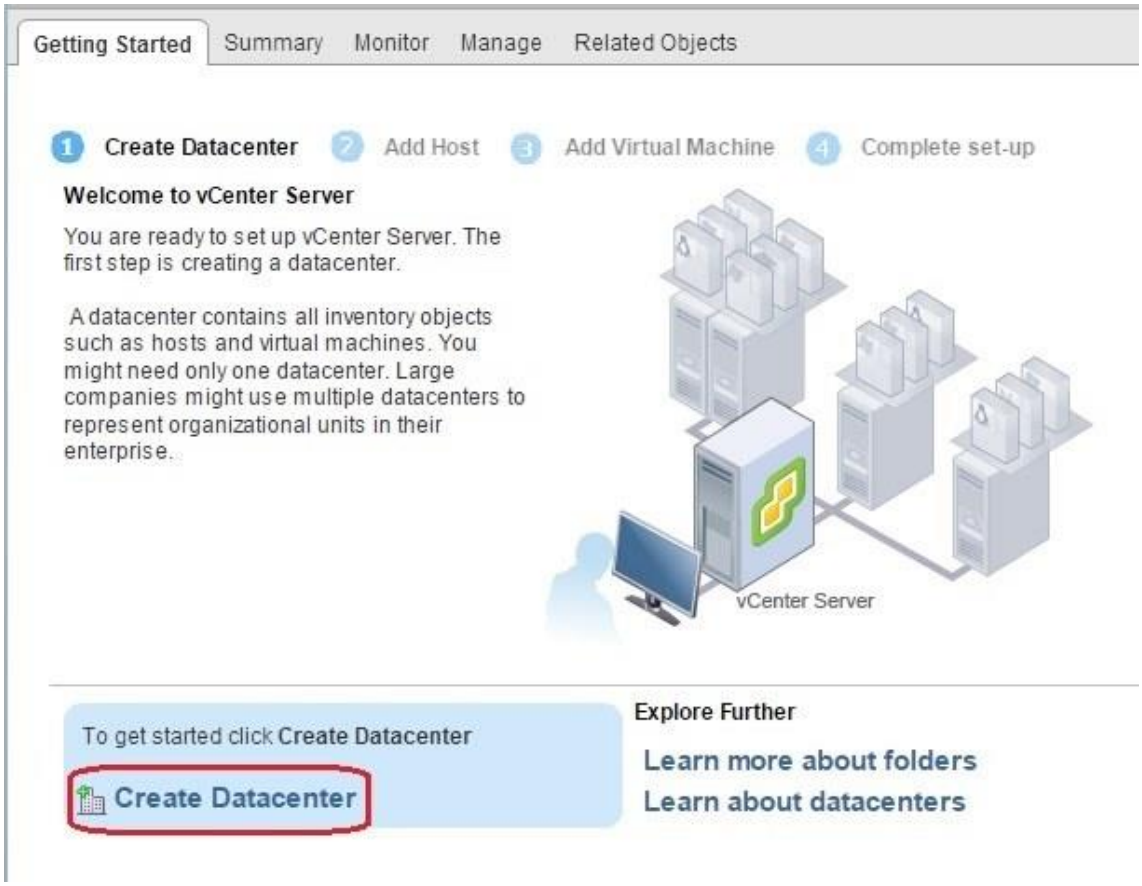

Figure 1.3 vCenter Server [1]

1.1.4 NSX

A network virtualization platform where virtual networks are programmatically created, provisioned and managed by utilizing the underlying physical network, called as NSX. When a VM is moved to another host, its networking and security services move with it. And when new VMs are created to scale an application, the necessary policies are dynamically applied to those VMs as well.

It allows organizations to implement virtual networks without interfering with existing applications and network configurations. With this network can be configured in very less time. Figure 1.4 shows the NSX Architecture.

Virtual network provides logical network services – logical switches, logical routers, logical firewalls, logical load balancers, and logical VPNs to connected workloads. These network and security services are delivered in software and require only IP packet forwarding from the underlying physical network.

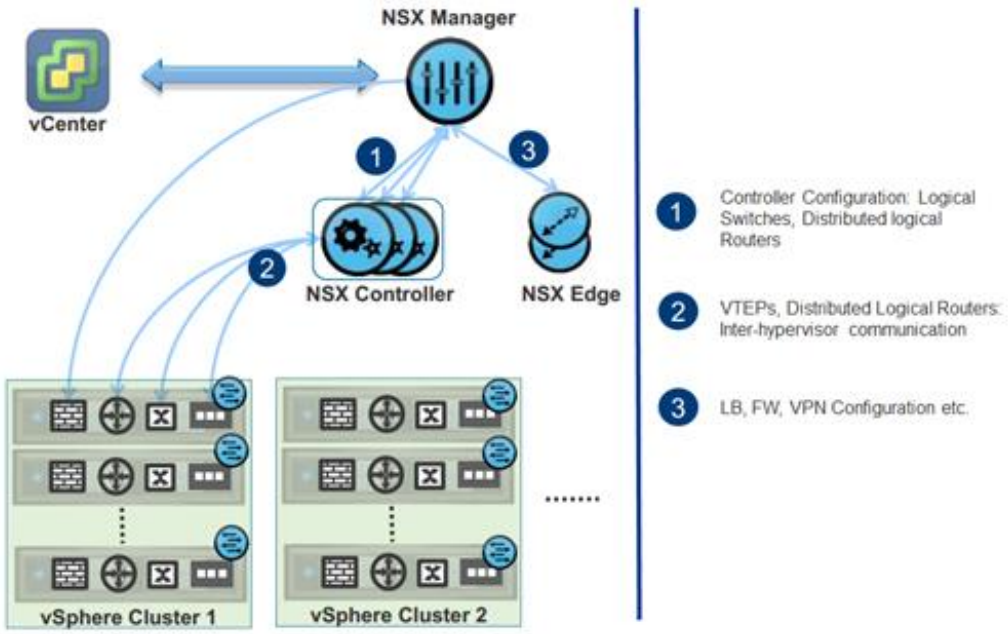

Figure 1.4 NSX Architecture [2]

1.1.5 Inventory

Inventory service provides data to other verticals about the environment; everything in the datacenter that is external to NSX and that NSX needs to interact with. Information for that can be collected through VC or through agent running on hosts and are provided to NSX management plane. Inventory contains items that are either discovered dynamically or pre-existing items that are registered by administrator. Some of the main objects that are identified are hosts, virtual machines and virtual network interfaces.

The inventory service is a part of NSX Manager Node. It is installed along with NSX manager node. The inventory service has an agent on the host that will be installed as a part of NSX bundle. Agent is connected to manager through MPA (management plane agent).

In case of NSX deployment, virtual machines and virtual network interface of distributed set of ESXi /KVM host needs to be discovered and persisted. But for on premise deployments, when number of host grow than it is difficult to persist all of the attributes of required set on NSX manager. So the inventory data should move to cloud. Figure 1.5 shows the components of Cloud Controller. Cloud controller is an appliance that is used to transfer data from on premises storage to cloud storage.

Most of inventory service data is ephemeral since it doesn't need the data and can collect it if anyhow lose it. Host and virtual machine identifiers are disk persisted but their properties are ephemeral.

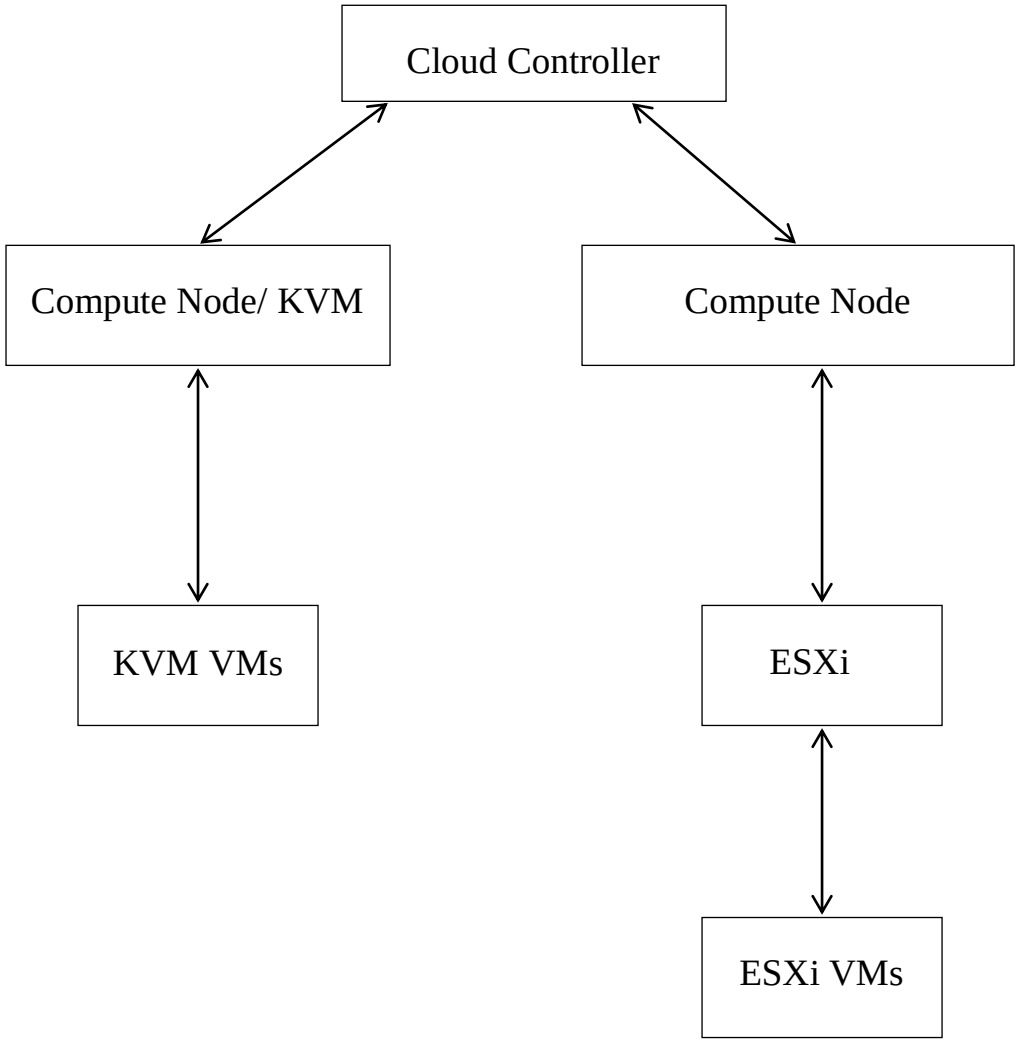

Figure 1.5 ESXi/KVM [3]

The discovery agent is a host-based agent that reports the VMs and VIFs available on the host.

It has the following designing goals:

1. Create an agent as part of MPA bundle that is installed on host.
2. The agent will be supported on ESXi and KVM.
3. Create a listener module in MP that gets the data and uses Object CRUD component to insert the data.

4. Entire host data will need to be reconciled in the following cases.
    a. NSX cluster recovers from the failure/shutdown and we need to re-discover all the VM/VIF data.
    b. Host agent recovers from failure/shutdown.
    c. Host agent does not receive acknowledgement for a certain period of time.
    d. Host has too many messages in queue and has to discard incoming notifications; it will clear the queue and ask manager for full sync request.
5. The agent will be installed as part of MPA bundle.
6. DA communicates with the management plane to send the data. The protocol is explained in later section.

Figure 1.6 shows the inventory data flow diagram. It shows various components that make up the inventory system. It comprises of agent, which contains data collector and agent queue manager. And on cloud side the manager is responsible for getting data.

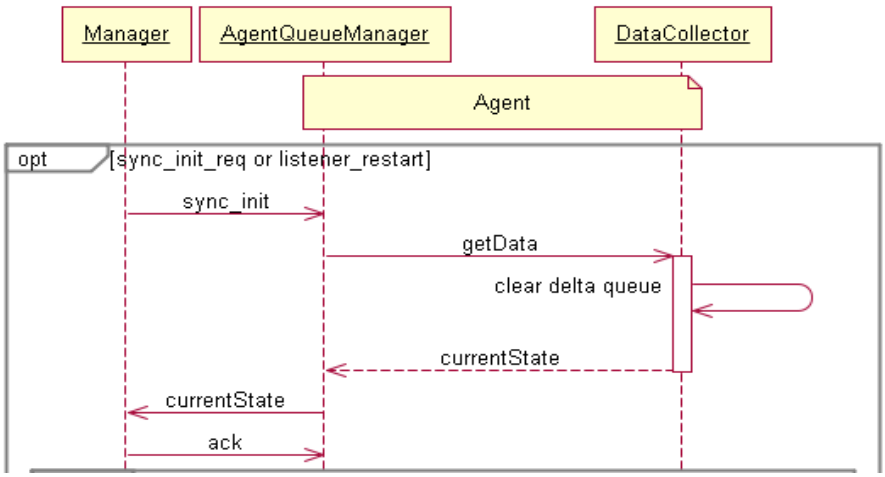

Figure 1.6 Inventory Data Flow Diagram [4]

This is the basic structure of inventory system in cloud environment. All the components are later described in detail.

IDA message sent from MPA to MP comprises of following information:

1. Host contains host id, host entry and interface. Host entry identifies the host properties and their values. Interface specifies interface type (Physical, Virtual), connected_switch etc.

2. Virtual Machine contains VmIdentifier and VmEntry. VmIdentifier identifies the VM name (ESX, KVM or HyperV) and VmEntry contains VmPropertyKey and its values.

3. Virtual Network Interface contains id, Vni entry, VmIdentifier and IP address information. Vni entry contains Vni's properties which define its states.

1.2 Chapterization

Rest of the thesis is organized in following manner:

Chapter 2 describes the earlier work that is done in the field of NSX inventory management. It explains the importance of inventory and its management.

Chapter 3 describes the existing system in detail. It explains all the components of inventory and contains the description of protocol.

Chapter 4 describes the gaps in existing system and then gives the description of the problem statement.

Chapter 5 discusses about the methodology. It describes techniques used to solve the problem and defines the proposed model.

Chapter 6 discusses about implementation. It describes all steps performed to solve the problem. It contains the architectural design and experimentation.

Chapter 7 focuses on the results obtained after the proposed algorithm was implemented and their analysis.

Chapter 8 discusses the conclusion and it is followed by future work of this proposed system.

1.3 Summary

This chapter gave the overview of virtualization and various products that form a part of virtualization platform. It describes basic functioning of various Vmware products.

Also it described the inventory service provided in this environment. After this it described the way the thesis report has been organized giving a brief description of each chapter.

CHAPTER 2

LITERATURE SURVEY

This chapter describes the research work done in the field of inventory data management and cloud environment. It helps to understand the importance of having the inventory services in any organization and also the need of managing this inventory data efficiently.

Every organization requires inventory for proper functioning of its activities. It links production and distribution processes together. In inventories, the investment is the most important part of current assets. Hence it is required to have proper management and control of inventories. The goal of inventory management is to make sure that the materials are available in sufficient quantity and to reduce the investment in inventories.

Deepti Theng and K. N. Hande [5] explained that “Cloud computing is the set of distributed computing nodes. The distribution of virtual machine (VM) images to a set of distributed compute nodes in a Cross-Cloud computing environment is main issue considered in this paper. This paper will be dealing with the problem of scheduling virtual machine (VM) images to a set of distributed compute nodes in a Cross-Cloud computing environment. To address this problem, an efficient approach for VM management is proposed and implemented in this paper. The result will be used as an effective scheduling guidance for VM scheduling on cloud computing as well as cross-cloud computing environment”.

Fildes and Charles explained that "Most companies evaluate the productivity of their inventories through such yardsticks as inventory turn, gross margin return on investment, gross margin return on square foot and the like. These are all valuable tools in assessing inventory productivity, but they are all limited by the fact that they use inventory at cost as the cost basis in their analysis. The true cost of inventory extends far beyond just inventory at cost or the cost of goods sold. The cost of managing and maintaining inventory is a significant expense in its own right, but the true cost of inventory doesn't even stop there. The full cost of inventory, in fact, is

actually buried deep within a number of expense items below the gross margin line, almost defying any executive, manager or cost accountant to pull them out, quantify and actually manage them" [6].

Shial and Majhi [8] suggested that for similar kind of files De-duplication works fine. Selecting the most suitable chunk size may result in production of maximum number of chunks, which in turn results in higher probability of getting maximum number of similar chunks. So finding best chunk size is a challenge in itself and it differs depending on the type of files being used for synchronization. Chunking method works well over all file synchronization technique. Apart from other file synchronization methods, Intellisync needs to setup client server architecture for the same.

Golle and Staddon (2005) presented Secure Conjunctive Keyword Search Over Encrypted Data [10]. A user stores an encrypted documents like e-mails, these types of settings on untrusted servers are done by them. So by this, for retrieving a specific document user needs to satisfy particular search criterion. After successful matching, user gives a server an authority that permits the server to spot those specific documents. Work here has generally fixated on look criteria comprising of a one watchword. On the off chance that the client is really keen on reports containing everything about watchwords (conjunctive catchphrase seek) the client should either offer the server abilities for everything about watchwords independently and depend on a crossing point count (by either the server or the client) to decide the right arrangement of archives, or on the other hand, the client may store additional data on the server to encourage such hunts.

Williams (2003) announced that a key component that isolates fruitful organizations from organizations that are not as beneficial is the means by which stock levels are resolved. An organization should first pick up a comprehension of where and how much stock is in the framework to viably decide fitting levels of stock expected to support generation and not have overabundance stock. This information can be get by breaking down what kind of interest the organization is encountering [11].

Bellare and Canetti [13] introduced Keying Hash Functions for Message Authentication 1996. In case of Internet applications and protocols, MD5 or SHA types cryptographic hash functions for message authentication has become a common

approach. They are very simple to implement but these mechanism supports ad hoc techniques that have less security analysis or have less secure. It has built new schemes of message authentication supported a cryptographic function. There are some message authentication schemes like NMAC and HMAC, tested to be more securing because they have some hash functions, which has affordable cryptographic strengths. Moreover it represented, in a very quantitative means, that the schemes retain almost all the security of the underlying hash function. As a black box, they use hash function or its compression function, so that mostly used or likely available library code or hardware will be fixed to implement them in a very simple way and changing the internal hash function is properly supported.

Organizations that arrangement with seaward providers sees stock levels in an alternate way. When managing seaward providers, organizations need to consider the travel time. A shipment from China requires 30 - 45 days from the time it leaves the dock in China until the point that it touches base at the dock in the United States. The 30 to 45 days incorporates the time that it takes to get the watercraft over the sea, the time required to dear traditions, and the travel time to convey the item from the port to the end client. Realizing that there is a 30 - 45 day lead-time, organizations need to hold in any event that much stock. Sometimes, organizations will hold 60 days of stock in the event that some disastrous occasion happens, similar to the vessel sinks, or the cargo is stolen. Organizations that hold abundance stock consequently experience stock conveying costs. Keeping in mind the end goal to figure stock conveying costs, an organization should first comprehend what is incorporated into the condition.

As indicated by Tersine (1982), conveying costs incorporate capital cost, charges, protection, taking care of, capacity, shrinkage, out of date quality, and weakening. The standard scope of yearly holding cost is 20 - 40% of the stock venture [16].

LaMacchia (2004) [19] additionally revealed that the cost to store stock for the most part runs 20-40% of the cost of merchandise sold (COGS) every year. Organizations need to comprehend the effects of holding abundance stock and start intends to lessen stock. Having exact stock records can enable associations to better comprehend stock needs, which will take into consideration diminished stock levels.

2.1 Summary

In this chapter literature survey was done. The research work done in the field of inventory management was studied. It helped to figure out the advantages of having an inventory system in cross-cloud environment.

CHAPTER 3

EXISTING SYSTEM

This chapter focuses on the detailed study of existing system. Therefore entire inventory structure is described in it. It describes the protocol that is used for transferring data from premises to cloud. Also it identifies the gaps in existing system and the fault scenarios, which trigger them.

3.1 Inventory System in Cloud Environment

In cloud environment whether it is on premise or public cloud there is a need to send inventory data related to the objects that already exists. This is done because it is not possible to save the entire information on premises. For this entire data set, which includes virtual machines and network interface information is sent to the public cloud.

The inventory is composed of various components which together make its working possible. These are:

1. Discovery Agent
2. Data Collector
3. Data Sender
4. Management Plane Agent
5. Management Plane

These components together are responsible for performing inventory operations. The working of these components is described below in detail.

3.1.1 Discovery Agent

The discovery agent is a host-based agent that reports the VMs and VIFs available on the host. DA (discovery agent) communicates with MPA (management plane agent). DA runs on the host.

First of all DA is started and its properties are loaded. After this it will get information about the host, which can be ESXi, KVM and hyper-v. And then data collector, data sender and a queue are created for communication between DS and DC. Also communication handler is created. MPAClient is also initialized and connected. Whenever DA has to be shut down, first stop DC, DS and MPAClient.

Discovery agent has the following designing goals:

1. Create an agent as part of MPA bundle that is installed on host.
2. The agent will be supported on ESXi and KVM.
3. Create a listener module in MP that gets the data and uses Object CRUD component to insert the data.
4. Entire host data will need to be reconciled in the following cases.
    a. NSX cluster recovers from the failure/shutdown and we need to re-discover all the VM/VIF data.
    b. Host agent recovers from failure/shutdown.
    c. Host agent does not receive acknowledgement for a certain period of time.
    d. Host has too many messages in queue and has to discard incoming notifications; it will clear the queue and ask manager for full sync request.

The agent will be installed as part of MPA bundle.

3.1.2 Data Collector

DC (Data Collector) is host specific. So every time DC is created it is verified that a specific host is valid. It will collect data for every host and push it to the data queue. DC can be in 2 states:

1. Blocked state - when any connection breaks
2. Sync init state - pushing data about host into queue

Initially isfullSync is set to false. Get full sync data and set SyncInitState to false and set fullSync to true.

If DC is in blocked state it will wait. If getSyncInitState=true, it indicates dc is going to send full sync and it is set to false. If getSyncInit=false, it will send full sync.

3.1.3 Data Sender

DS (Data Sender) pulls the data pushed by DC into the queue. It is then sent to MP by making use of MpaClient. First it will check if there is any message on queue. If not it keeps on waiting till some message comes. After this it will get DS state. It can have 2 states: waiting for data and DO_SYNC_INIT.

1. Waiting for data - DS enters this state after receiving ACK from MP after receiving data (full sync or update). So in this case DS is expecting only updates. First it will check DC message type. DC message type can be:
    a. DMT_INIT_START- this means something went wrong because of this full sync needs to be done. So state is changed to WAITING. DS will again raise SYNC_INIT_REQ.
    b. DMT_INIT- ignored
    c. DMT_INIT_END-ignored (will break in both the cases because this is some failure case)
2. DO_SYNC_INIT- full sync is expected here. DC sends all the data in between DMT_INIT_START and DMT_INIT_END message
    a. DMT_INIT_START- clear if we already have some messages.
    b. DMT_INIT- actual objects (info about VMs, VIFs) are sent. All the messages in between start and end messages are sent.
    c. DMT_INIT_END-Signifies we have reached end of info. Now we send entire data to MP
    d. DMT_UPDATE- No operation is performed

3.1.4 IDA message

IdaMessage is the message sent from MPA to MP. It contains three things:

1. Message Type

2. Inventory Object
3. Ida command

Message Type defines the type of message. It can be:

1. MT_INIT
2. MT_UPDATES
3. MT_COMMAND

Inventory Objects contain information about the object like object type (Host, VM, VNI), Change type (CT_CREATE, CT_MODIFY, CT_DELETE), Hosts, VMs and VNIs.

Ida commands are of following types:

1. SYNC_INIT_REQ: When DA communicates with MP, it sends this message.
2. SYNC_INIT: MP reverts back for requesting full sync data
3. ACK: After getting full sync data. MP sends this ACK to DA.

3.1.5 Backup and Restore

There is no impact of backup/restore on inventory for the following reasons.

1. Inventory data is ephemeral. When the cluster restarts, the agents will connect and give the latest data to the management plane as part of sync operation.
2. Inventory agent uses MPA to connect to management plane. Any changes on management plane that was lost, as part of restoration should not affect inventory agent directly. They might affect MPA.
3. The persistent data of inventory will be backed up and restored as part of the regular backup and restore operation, and what is lost is obviously lost same as other verticals. There is no special case for inventory identified.

3.2 Inventory Management System Protocol

The protocol defines how data is being sent from on premises to cloud. Multiple components together make this working possible. DA (Discovery Agent), which sits on ESXi host, will send SYNC_INIT_REQ to MP (management plane) through MPA (management plane agent), whenever there is connection between MP and MPA. When MP notifies that it is ready to receive full sync, discovery agent will send SYNC_INIT. At this point entire host inventory information about virtual machines and virtual network interfaces is sent to MP. From now onwards-entire data will keep on syncing from on premises storage to cloud storage. Figure 3.1 describes the protocol of inventory management system. It shows how the data is being sent from discovery agent to management plane. Various messages sent at different point of time are shown in the figure.

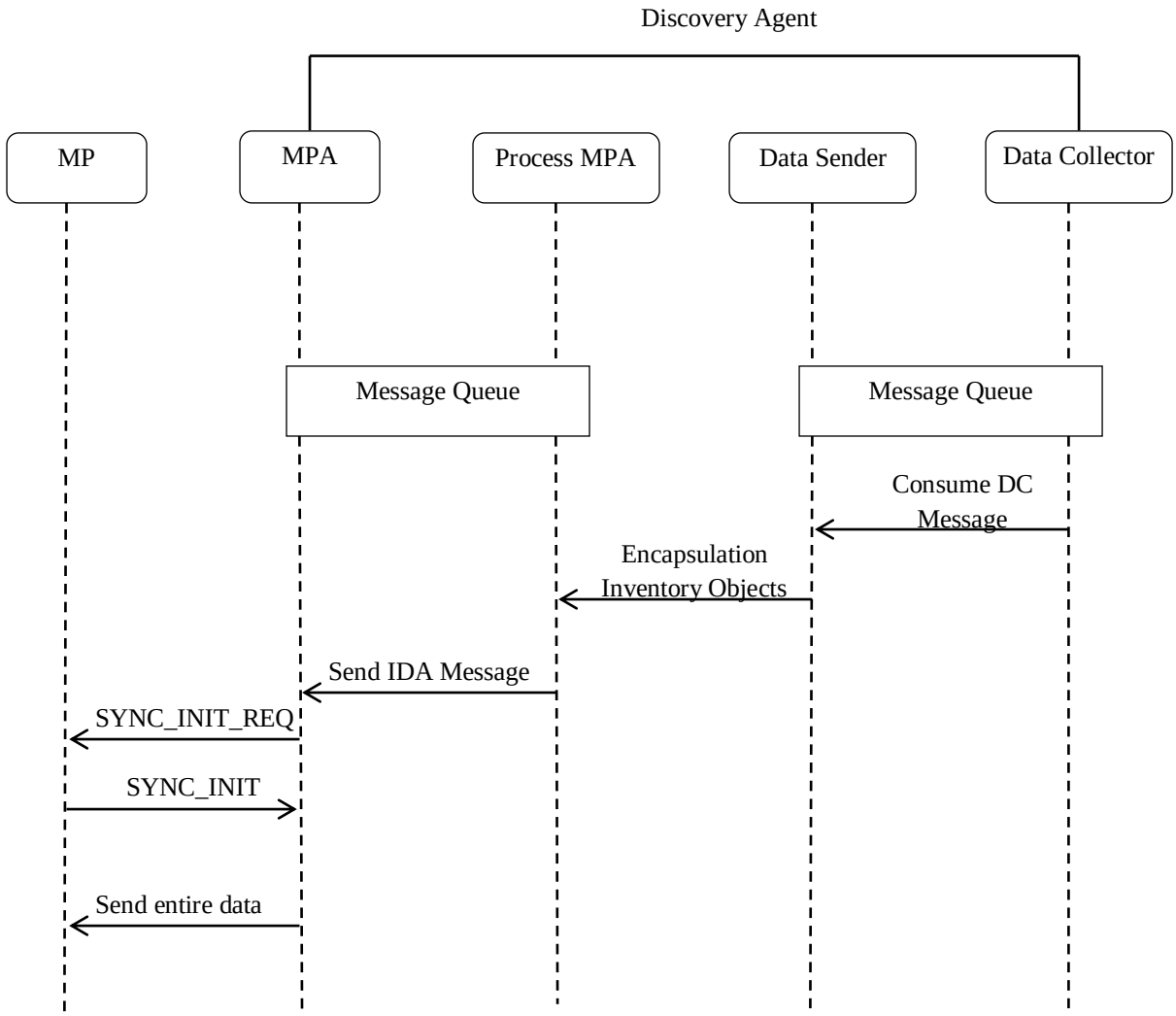

Figure 3.1 Inventory Management System Protocol [13]

3.3 Full Sync Scenarios

There are some full Sync scenarios (Fault Scenarios), which trigger re-populating entire information. Some of these scenarios are discussed below:

1. Hardware and software Refresh:

   As the innovation develops, there is noteworthy change in server's life expectancy, yet even with different choices accessible for expanding the server's life expectancy; there is still need to buy equipment. There comes a point, when server equipment is essentially excessively old, making it impossible to perform and again question is when might you know when it's truly time. To think about the new equipment buy, the key is to distinguish that, when expanded execution, vitality productivity prerequisites will legitimize another equipment buy [13].

2. VM Migration:

   Conversion from physical server to virtual server (VM) is called as physical to virtual migration. For performing physical to virtual machine migration, we need to copy the physical server's bits to virtual disk, install drivers and modify the guest to support these drives.

   Workload keeps changing in servers. There is always a sudden spike of requests i.e. once in a month. At that time, it is not possible to provision additional servers manually.

3. Unplanned datacenter downtime:

   a. Human error and poor infrastructure capacity management.

   b. Poor maintenance and lifecycle strategy.

   c. Substandard data center site selection and risk mitigation.

   d. Hardware/Server failure

   e. Servers, which are presented in the datacenters, are always running and continuously heating up, this is the reason for h/w failure. There are some more reasons for equipment failure, like if temperature increases from its

fixed value than it decreases the server's performance and might be reason for datacenter failure.

4. Power Outage:

   The reason for power outage is uninterrupted power supply (UPS). It can happen from battery failure or from the excessive power draw which reaches beyond the predefined limits.

5. Host Goes Down:

   The reason of host breakage is the network traffic between the host servers and SAN (Storage area network which is the location of VM file).

3.4 Summary

In this chapter inventory system in a cross-cloud environment was described in detail. All the components that together make up the inventory were discussed. After that the protocol for transferring data to cloud was described. Then it discussed the fault scenarios, which trigger repopulation of data.

CHAPTER 4

PROBLEM STATEMENT

This chapter discusses the gaps found in the existing system and defines a problem statement. It identifies the goal of the work to be performed further.

4.1 Gaps in Existing System

The inventory state information keeps on changing from time to time. Sometimes new objects are added or existing objects are updated. Multiple hosts are responsible for making these changes. Thus the inventory information on cloud needs to be updated from time to time. So again entire data will have to be sent.

Also there are some other cases where there is need to send complete data:

1. Host recovers from failure
2. Cluster recovers from failure
3. Connection between MP and MPA breaks

In these situations also there is a no way to find out how much data has been already sent. So again entire data has to be sent.

This repopulation of entire data set again and again adds to the bandwidth cost. That is why it is not an effective approach. So there is need to find out a way, which reduce the amount of data being sent across cloud.

4.2 Problem Statement

With the extreme grown in technology, the amount of data that belong to an organization is many folded. It is practically impossible to persist entire data on premises. So many organizations are turning towards cloud. So the focus is on cross-cloud environment where NSX manager that manages both on premise and public cloud workflow is deployed on premises of an enterprise customer. In such a

situation, data must be sent to the public cloud environment. But there are certain fault scenarios, which require whole data to be sent again and again to cloud.

4.3 Objective

The main objectives that are identified in this thesis are:

1. Studying the full sync scenarios.
2. Finding the current size of inventory information for a typical NSX deployment.
3. Propose a solution to effectively reduce the inventory data exchange.

4.3 Summary

This chapter identifies the gap in the existing system. It describes in which ways it was not efficient. It defines the problem statement and focuses on defining the main objectives of the new system.

CHAPTER 5

METHODOLOGY AND PROPOSED WORK

This chapter focuses on the methodology, which describes the system that will be used to carry out the experimentation. It also discusses the proposed solution to enhance the existing system by solving the problems that are present in it.

5.1 Methodology

Inventory services normally depend on compute managers such as VC to collect information about objects in the compute stack. It needs to access some basic objects and properties. The objects that are identified are:

1. Host
2. VM
3. VIF

While the data collection cannot be done through compute managers, the idea is somehow have this data collected and provide this information through the standard inventory interfaces.

Following use-cases have been identified.

1. Admin should be able to add hosts to the inventory using NSX Manager API. They should be able to update or delete the said hosts as well.
2. Inventory should have data about VM and VIFs on the hosts.
3. UI needs information about the Mac address of the VIF, name of the VM and its location that is the host.
4. There should be a unique identifier for every object present in the inventory.
5. Data from multiple hosts can be sent to the inventory.

Figure 5.1 shows the inventory services.

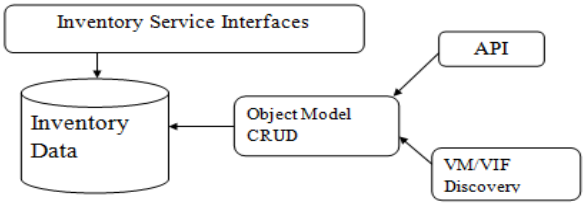

Figure 5.1 Inventory Services

5.2 Object Model

It describes the basic inventory structure used identify the objects present on a particular host.

5.2.1 Inventory Objects

The objects available in the inventory are:

1. Hypervisor Info:
    a. Id (Copy of External id): Used as an identifier by NSX.
    b. External id (Optional): As entered by admin. This is the UUID of the host as per the information on the host. Both ESX and KVM have this property. ESXi and KVM require this.
    c. Display name (Optional): As entered by admin.
    d. List<IP address of the host> (Ipv4/Ipv6): As entered by admin- used to identify the host in our system.
    e. Type: ESXi or KVM.
    f. Version: version of the software running on the node such as ESXi version or KVM version.
2. Host:
    a. Combine information from hypervisor info and Messaging Client Info (a shared object between different message clients).
3. VM:

a. Id (generated id): Generated internally by NSX.
b. External Id: This will be the combination of BIOS UUID and path of the VM. This is required to identify the VM in our system. Discovery agent will provide this.
c. Display name (Optional): As discovered by the agent.
d. Type: defaults to regular but edge vertical can mark the VM as an Edge VM.
e. Host id
f. List <VIF.id>

4. VIF:
    a. Id (generated id): As discovered by agent and same as external id.
    b. External id: Combination of VM id and device key (ESXi) or device alias (ESXi).
    c. Vm id
    d. Host id
    e. List<MAC address>: as provided by admin or collected from discovery agent.

5.2.2 Display Name

For host, this depends on whether user has entered and display name.

For VMs, this is discovered through a host agent.

For VIF, this is the combination of VM name and nic ordinal number or VIF id

5.2.3 Scale Targets

Data center consists of multiple hosts. Each host can have as much number of virtual machines. Virtual machines further have virtual network interfaces whose number can vary from VM to VM.

Scale targets for Avalanche are:

1. 10K VMs
2. 25-50K VIFs
3. 500 Hosts

5.3 Proposed System

In order to solve the identified problems, there has to be some checkpoint, which keeps track of the data that has been already sent. The proposed solution is to calculate checksum of data file. Initially the entire data file is sent. After that for all the updates being made on data file checksum is calculated and all the updates are stored in separate files. A separate file is maintained that contains all update files names and their corresponding checksums. Every time some update is made the latest checksum is sent. This checksum is compared with the checksum of file present on cloud. If this checksum matches it means file is up to date. If it does not match the checksum of file present on cloud is sent and it is compared to all the checksums and the point where it matches from then onwards the updates are sent. This eliminates the need to send entire data.

DA (Discovery Agent), which sits on ESXi host, will send SYNC_INIT_REQ to MP (management plane) through MPA (management plane agent) when there is connection between MP and MPA. When MP is ready to receive full sync, it will send SYNC_INIT. At this point entire host inventory information is sent to MP with the calculated checksum.

But from then onwards only checksums will be transferred rather than whole data. If checksum matches that means there is no need to send any data. It shows that the inventory information on cloud is up to date. But in case both the checksums do not match, it signifies that there are some updates, which have not been reflected on cloud storage side. Therefore those updates need to be identified. For this the checksum that has been calculated on cloud side which here is MP will be transferred to the MPA, where it will be compared. This process continues until whole data reaches cloud. Now there is no need to send whole file again and again.

Figure 5.2 shows the working of proposed system.

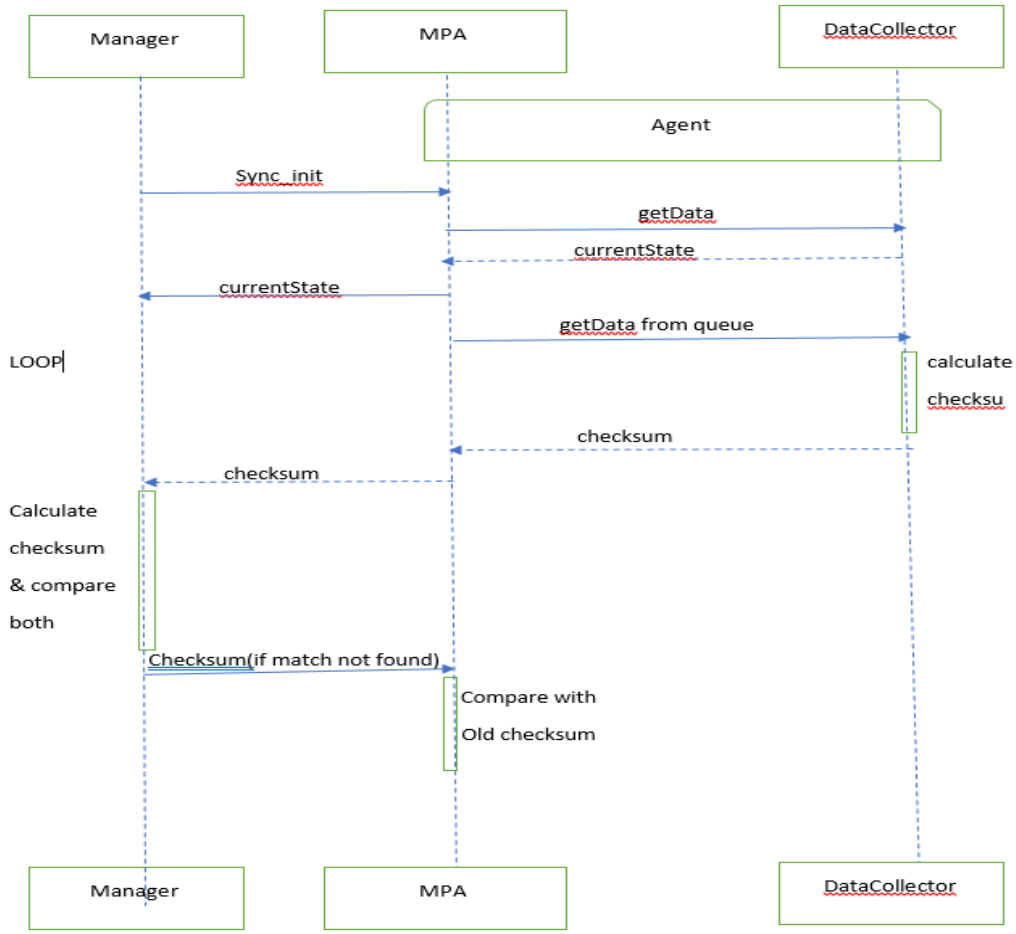

Figure 5.2 Proposed System View

The solution that has been proposed focuses on saving a checkpoint that will assist to keep track of the successful sync of the inventory information.

5.4 Summary

This chapter describes the methodology that will be used for implementation purpose. It describes the basic inventory management system setup that will be used to solve the existing problems. It also proposes a solution that will be used to solve the gaps that have been identified in the existing system.

CHAPTER 6

IMPLEMENTATION

This chapter focuses on the implementation of the proposed system. It describes the architectural design of the client server application that has been built to implement the system. It also describes the working of the proposed system.

6.1 Application Design

Client Server Architecture has been used to test the proposed solution. Client side does the work of discovery agent. It has access to the file that contains information about the VMs and VIFs. All the updates to this file are made on client side. Once the changes have been reflected on the original file, they have to be applies to file on server side. So there is a need to save these changes separately so that there is no need to send the entire data. On server side there is only one file that will store the updated data. The implementation has been done in java. It comprises of following classes.

6.1.1 File Reader/Writer:

Various operations can be performed on the file that contains inventory information. The main operations are CRUD (Create, Read, Update, Delete). These operations are made based on unique id called the external ID.

1. Create: It is basically a write operation that is used to make new entry, which can be either a VM or VIF. Entire information of the particular object needs to be specified. But in case the external id entered already exists, the entry will not be made.

2. Read: A particular VM or VIF properties can be read by using the external ID.

3. Update: For performing update operation, get the ID of VM/VIF and make changes according to the requirement. If the ID entered does not match nay existing object then message will be conveyed.

4. Delete: For performing delete operation, get the ID of VM/VIFs and delete the particular entry corresponding to that id.

Once the update is requested the changes are reflected in the original file. But along with that requested updates are saved in separate file so that only those can be sent to the server.

Figure 6.1 shows the workflow for performing CRUD operations on a file. Initially the file that contains the original inventory system data is read. The basic CRUD operations can be performed on this file. Once the operations are performed successfully the original file is re-written. Along with it the updates that have been requested are stored as a separate file. Naming of this new file is done in sequential order by making use of timestamp.

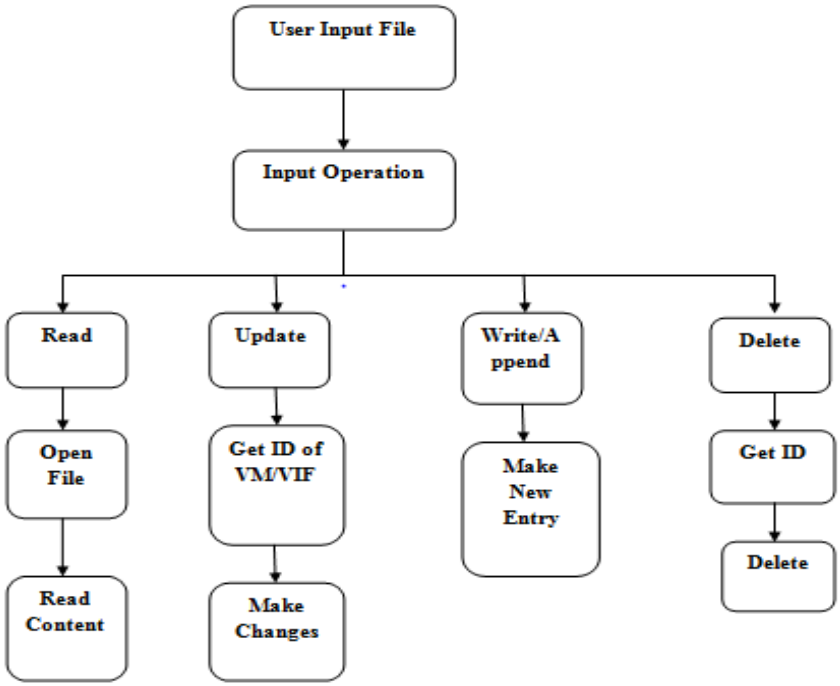

Figure 6.1 Workflow for Performing CRUD Operations

6.1.2 Checksum

Various checksum algorithms are present but MD5 algorithm is used here. Once the file is updated it is sent to checksum class. Here the checksum of the sent file is calculated. A separate file is used to store the calculated checksum value and name of file where the update is stored. This is later used to keep track of the data already sent

to server. Same algorithm is used on the server side to find the checksum of the file present on it.

Figure 6.2 shows the working of checksum class.

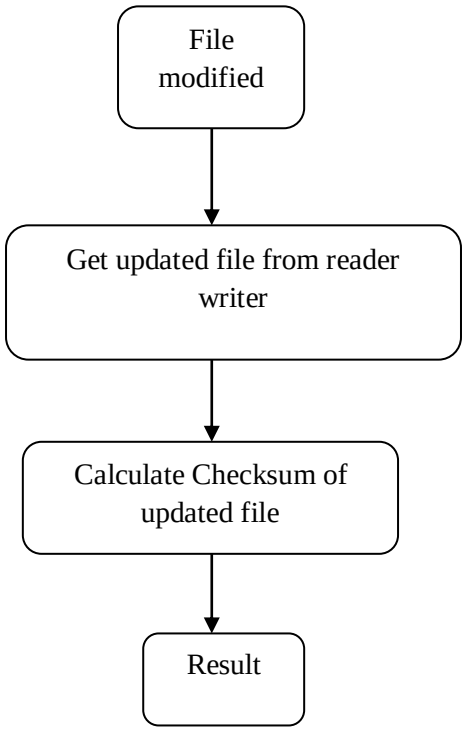

Figure 6.2 Flow Diagram for Checksum

6.1.3 Http Client/Server

If there is no file on server, full data file is sent from client. After that whenever any update is made on client it will send checksum of latest update performed. On server side, checksum is calculated of the file present on it. If the checksum received from client matches the calculated value on server side, it implies that server is up to date, there is no need to send any data file and server will send success response to client. If the value does not match, server will respond with its checksum. The client maintains a file, which contains all the checksum values in sequential order of the timestamps with corresponding update file. It will start comparing the value sent by server to it and the point where it matches some previous checksum value, client get to know how much data has been successfully received by server and it will send only rest of the updates.

Figure 6.3 shows client server system using http protocol.

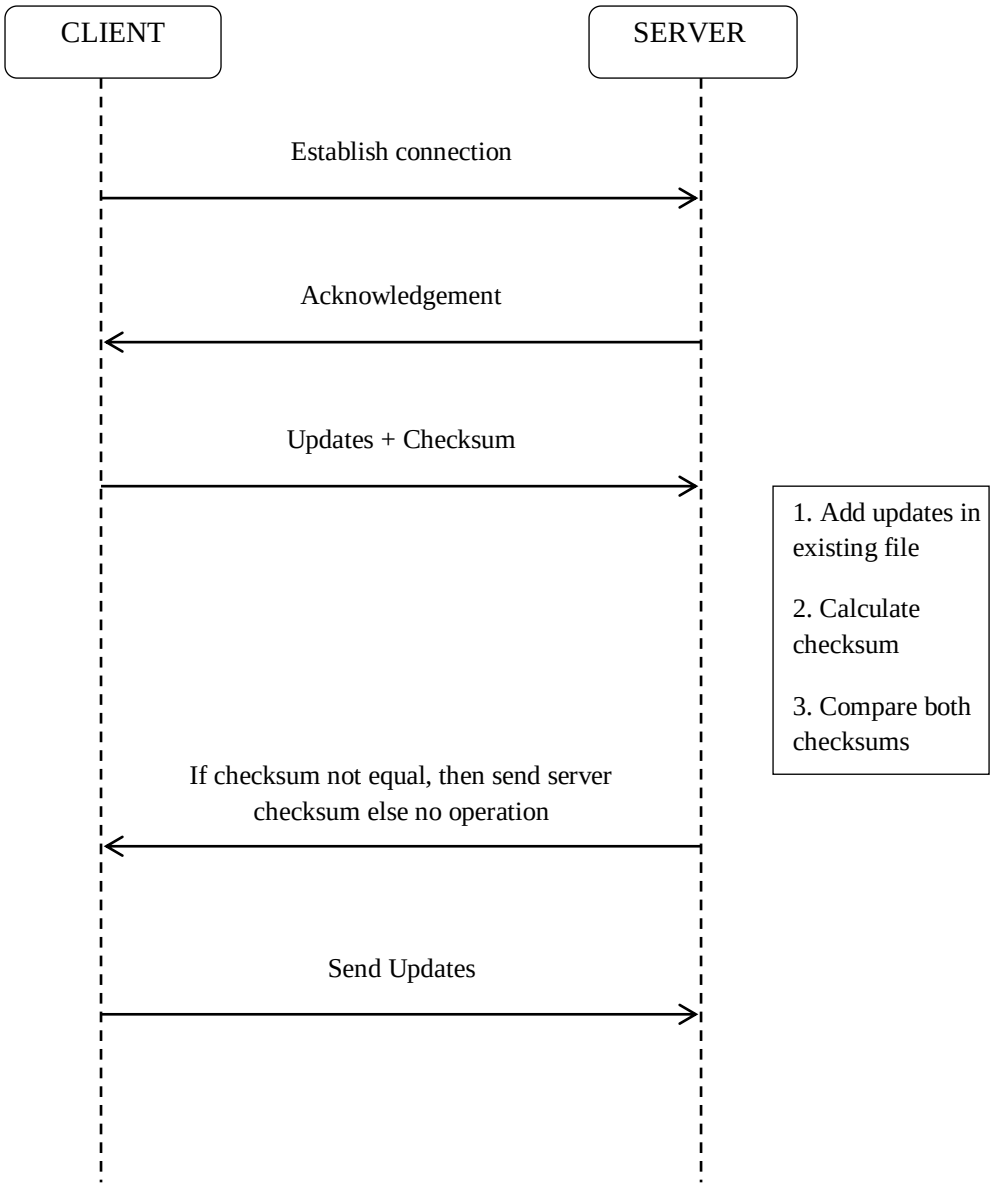

Figure 6.3 Client Server Workflow Diagram

6.2 Experimentation

This part describes the actual working of the client server application that was developed to test the propsed solution. The application design has been done in spring framework. Here the client and server are both present on same machine. Step wise description of the entire working has been given below.

1. First make a client-server connection. For sending data, server should be started. Server should be started from a port that is free and client should be listening to same port.

2. Initially there is no file on server so send original file to it. Hence client and server have same file. After this there is silent communication between client and server

   Figure 6.4 shows the vif.json file that needs to be sent on server side. This is the original data file.

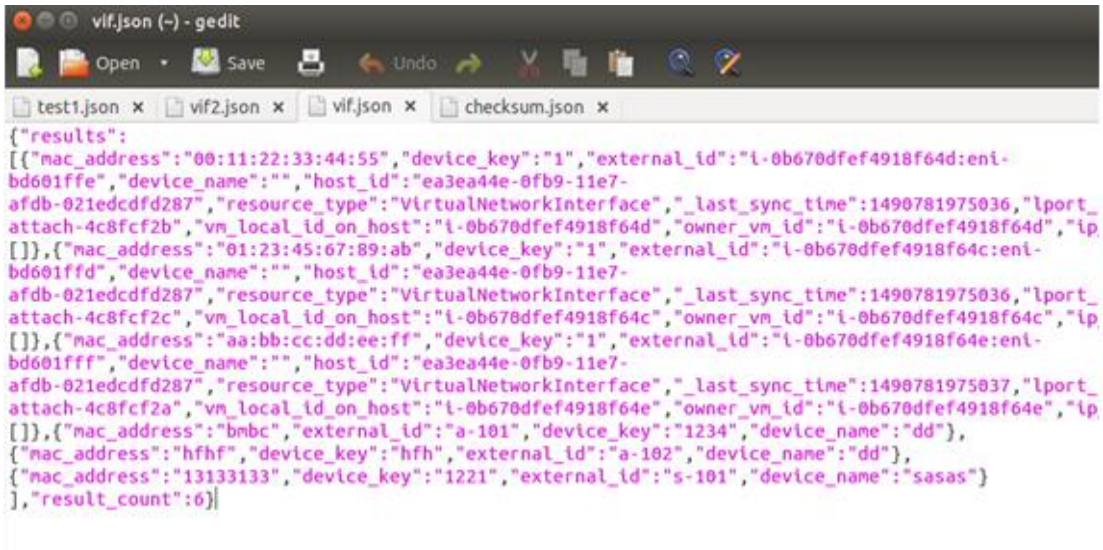

Figure 6.4 Original Data File

3. Whenever any update (Add, Update, Delete) is performed at client side a file is maintained on client side, which contains only the updates that have been performed currently. Figure 6.5 shows the test1.json file that contains the update operations that are performed by client on the original file.

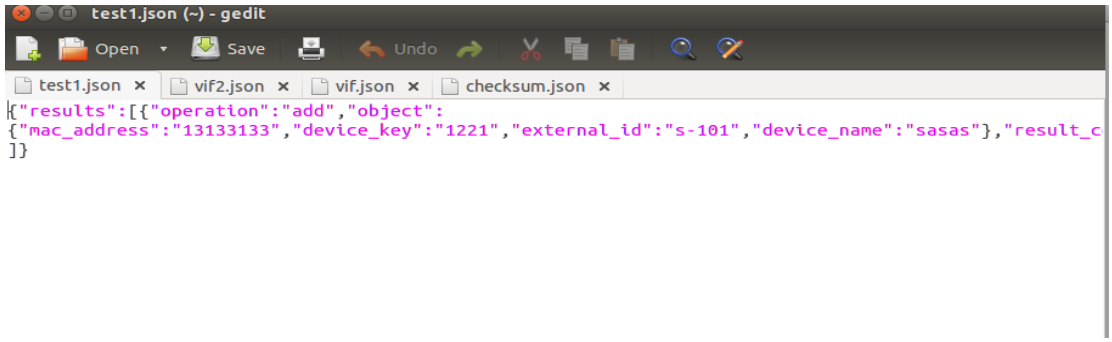

Figure 6.5 Modified Data File

4. A file is maintained on client side containing checksum of file along with corresponding update file. Here vif.json file is already there on client side and remaining are updated file's checksum. This file is used to compare the

checksum sent by the server when client and server checksum does not match. Updated file contains new operation that is performed by clients.

Figure 6.6 shows the checksum.json file that contains the checksums calculated by md5 algorithm.

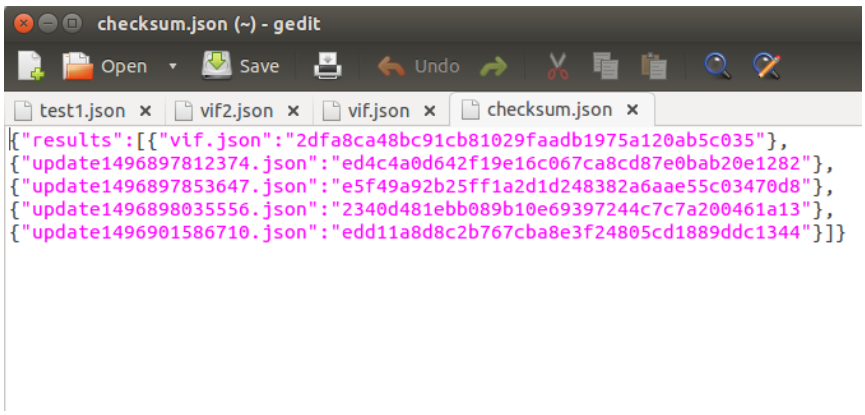

Figure 6.6 File Containing all Checksum Values

5. Client will send the checksum of updated file to server and server will receive that checksum and will match this with the checksum of recent file available on server side. Figure 6.7 shows the client's output after performing new update operations.

Figure 6.7 Client Side Output

6. If both checksums the one calculated on client side and other calculated on server side are equal than there is no need to send updated file from client to server. Server will respond with a success message.

7. In case checksums do not match, server responds to the client by sending the checksum of the file present on it. Client needs to check from file which checksum (present on client side) it matches, and send the remaining updates to server side. In this way client gets to know how much data has been saved successfully on server and send only rest of updates. Those updates are then applied to file present on server side.

   Figure 6.8 shows the updated file after performing new operations.

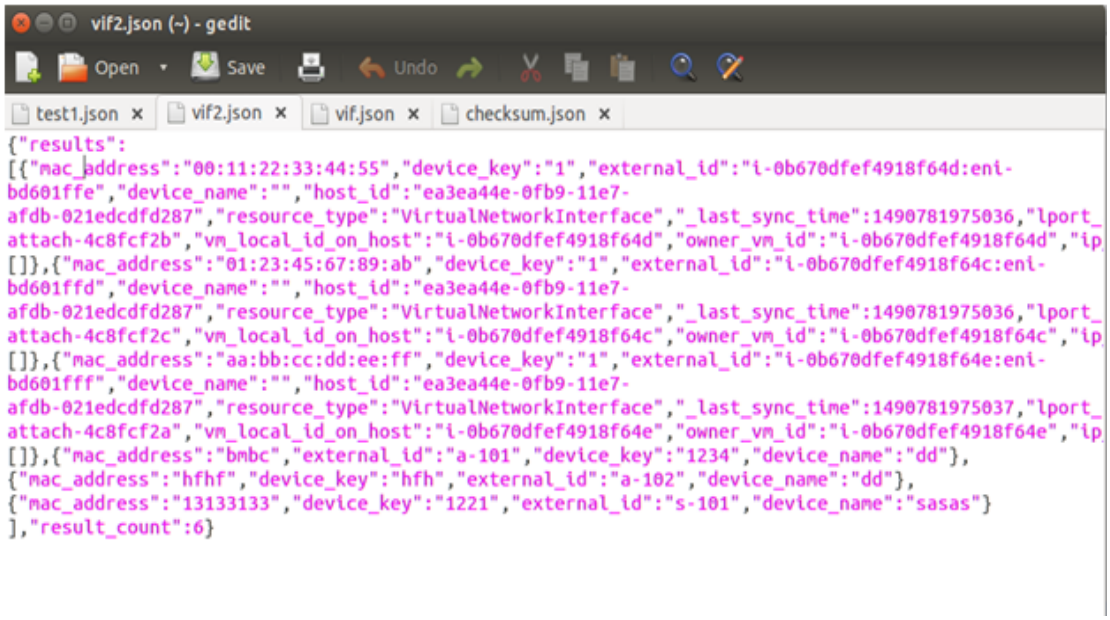

Figure 6.8 Updated File on Server Side

These are the solutions for the scenarios where current protocol fails and needs to send full data again and again to the cloud.

1. Hardware/Server Refresh: Sometimes, a server equipment refresh can unravel execution issues at the part of the cost of another server. Including CPUs or memory can essentially expand server execution. However all the hardware cannot be fixed by up gradation because not all the hardware is upgradable. There is always a need for new purchase.

2. VM Migration: Virtual machine migration accommodates for changing workloads automatically. It can manage to both additional workload and reduced workload. For the users of servers, a scheduled maintenance normally results in some downtime. Virtual migration helps to migrate virtual machines

from one host to another and brought back to the original server when migration is completed. If there is a case of unscheduled server downtime, virtual machine migration can also migrate from one host to another host. (This is the case of any random server fault). For disaster recovery, virtual machine migration can also be used. This process involves setti0ng up of similar resources in a disaster recovery site with high speed Wide Area Network links and specializes network connectivity equipment. Between primary server and the servers in the data recovery site, the virtual machines and their array Comparative to migration of operating systems and applications, migration of virtual machine is quite effortless process.

With the help of virtual machine migration, it is possible to migrate operating system from older server to newer one easily and without disrupting the services [4].

3. Unplanned Datacenter Downtime: This can be managed by buying better quality hardware from the known vendors.

4. Power Outage: Sometimes, a server equipment refresh can unravel execution issues at the part of the cost of another server. Including CPUs or memory can essentially expand server execution. Batteries should be tested at least quarterly for verification of correctness.

   Cyber-attacks may be a reason for power outage. CRAC failure has also become increasing common as densities rise.

5. Host Goes Down: The suggested solution for host failure is VMware high availability (HA). It helps by gathering all the VMs and hosts on which VMs present. HA first checks for failed host and then it can restart a VM, which is running on a failed host. HA can also check by checking the VMware tools, which are running on it. When any ESX/ESXi host (Host on which VMs are running) fails for any reason than VMs can also fail. VMware HA ensures that VMs, which are running on failed host, can run on another host (Means they should be capable of being restarted). ESX/ESXi host failure is primarily dealt by the VMware high availability and what happens with the virtual machines that are running on the host. HA doesn’t mean fault tolerance. These are two

different terms, HA checks availability for VMs, which failed because of host failed, and fault tolerance focuses on making system more fault tolerance.

6.3 Summary

The main focus of this chapter was to discuss the design of the client server application, which has been developed to test the proposed solution. It describes the various classes that have been used in this application. It discusses various outputs given by this application. With the help of these outputs the proposed system will be compared with the existing system.

CHAPTER 7

RESULTS AND ANALYSIS

This chapter is divided into two sections. First section describes the results obtained after the implementation of proposed solution. It discusses the results of existing and proposed system. Also it compares both the systems. Second section analysis both the systems that is existing and proposed.

7.1 Results

A client server application was developed. The client side performs all the CRUD operations and stores the checksum of files. Whenever some update is performed the data goes from client to server. The server side will store the data sent by the client. The size of the data being sent from client to server was calculated. The results were calculated for sending the entire data and for sending only the updates. This result was calculated for 512 hosts having 10000 virtual machines and 50000 network interfaces.

The size of message for objects is calculated using an existing Vmware tool. It finds the size of message sent from on premises to cloud. It gives the size of message for virtual machine, virtual interface network and the host.

The results of existing system as well as proposed system are described below.

7.1.1 Existing System

In existing system entire data is sent from premises to cloud. The reason was there was nothing that could point to the last successful sync. The size of message for one host is calculated and then it is used for calculating multiple hosts.

Table 7.1 shows the amount of data that has been sent by each VM and each VIF. This information was used to calculate the data sent by each host. After this size of message was calculated for all the hosts sending data.

Table 7.1 Size of message for existing system

Object	Number of objects	Size/object (in GB)
VM	10000	0.24
VIF	50000	0.38
HOST	512	35

Total size of message is 17958 GB approximately.

7.1.2 Proposed System

Here instead of sending the entire data only updates were sent. Size of message for a proposed system is calculated here.

Table 7.2 gives the description of proposed system. It shows the amount of data that has been sent by each VM and each VIF. This information was used to calculate the data sent by each host. After this size of message was calculated for all the hosts sending data.

Table 7.1 Size of message for proposed system

Object	Number of objects	Size/object (in GB)
VM	10000	0.02
VIF	50000	0.04
HOST	512	3

Total size of message is 1536 GB approximately.

The size of message reduced by 91.45 %. This result is very significant. As the amount of updates increase the size may increase a bit but still it is very effective. It will reduce the bandwidth cost of moving the data to the cloud. So there would not be a need to pay high expenses of sending large amount of data over cloud.

7.2 Analysis

This section describes the analysis of the existing system and the proposed system. It describes the difference in functioning of both the systems. Then the results obtained on performing the experiment are analyzed. It discusses the ways in which proposed system is better than the existing one.

The amount of data that has been sent by each VM and each VIF has been calculated. This information was then used to calculate the data sent by each host. After this size of message was calculated for all the hosts sending data.

This data was used to compare the existing and the proposed system. It showed very significant difference between both the systems. The size of message when entire was sent in every scenario was found to be very large. This was in the case of existing system. But when only updates were sent in case of proposed system the size of message was reduced. The organization where the data keep on changing can take the advantages of having reduced data.

7.2.1 Existing System

In existing system, whole data needs to send again and again whenever any crash happens because of hardware failure, power outage etc. also whenever any update is performed entire data is required to be sent from on premises to cloud. This increases the cost of whole data transfer. The reason for increased cost is hike in the bandwidth cost. When the number of hosts increases there is a huge increment in the amount of data being sent. It becomes very inefficient to handle this large amount of data. So because of these problems there was a need to find out a solution, which resulted in minimum repopulation of inventory data.

According to the results given in table 7.1 the size of message that is being sent in existing system is approximately 35 MB for one host. But as the number of host sending the data to cloud increase this message size many folds. For a typical NSX deployment, which can have up to 512 hosts, this size amounts to 17958 MB. Sending this much amount of data again and again is not at all cost effective. So there was a need to identify some other approach. So a new system was proposed where a checkpoint was used in form of checksum.

7.2.2 Proposed System

In case of proposed system, only updates are being sent that reduces the cost of whole transfer and amount of data. This system only saves updated file's checksum instead of whole file and that checksum needs to be maintained in both sides i.e. server and client side. In case of proposed system, whenever any crash happens, server and client both already have the last checksum details stored in a file. That is the reason of sending updates only instead of full sync. This minimizes the amount of repopulated data and hence will also help in reduction of the cost of sending the data to the cloud. This solution can server very useful for the organizations where the data keeps on from time to time.

The results of proposed system showed the reduction of message size in the proposed system. As depicted in table 7.2 the size of message for a single host was found out to be 3 GB approximately. Thus maximum size of message for multiple hosts can be about 1500 MB. Thus the size of the message in proposed solution was reduced by 91% approximately. In a data center having 500 hosts reduction of this much data is of great significance. It will reduce the overhead of sending mega bytes of data over and again.

7.3 Summary

This chapter gives the description of the results given by existing system and proposed system. It compares both the systems. After this analysis of the work has been done.

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

This chapter will discuss the conclusion of the work that has been done. Then the future work that can be done in inventory management system will be discussed.

8.1 Conclusion

In this thesis report, inventory system was analyzed. The main problem found was repopulation of the inventory data. So various fault scenarios were identified which lead to full data sync.

For performing experiment client server application was developed that was similar to the actual cross-cloud structure. The size of message has been calculated for sending the data in cross-cloud environment. This size was calculated for various inventory objects which are virtual machine, virtual network interface and host. The results were calculated for existing and proposed system. According to the proposed system, when only updates were sent to the server, there is significant reduction in the amount of data being transfer from on premises storage to cloud storage. It can further result in reduced cost of sending the data to the cloud storage.

The proposed system has the following limitations:

1. Only a limited set of objects and properties are supported.

2 The data from host may not be immediately available after a cluster restart, since at max scale, we would access data from hosts in batches to avoid flooding the message bus.

3 A newly added host may not have all data discovered immediately since other hosts might be queued up already for the first sync.

8.2 Future Scope

The identified limitations can be overcome in the future work that will be done in inventory system management.

While doing this work, MD5 algorithm was used to calculate checksum of files. The main aim was to reduce the amount the data, so simple algorithm was used to find checksum. This work can be further improved by using more efficient algorithm to do calculation.

This solution works for same checksum algorithm used on both the sides that is the client and the server. This makes system hardly coupled. In future it can be extended for different development platforms having no condition of using same algorithm on both the sides.

The information that was sent to server was stored in similar way as it was present on client. Therefore it was easy to apply updates. But there may be situation where data on client and server side cannot be stored in same format. So in this case data can be serialized on both the sides.

REFERENCES

[1] "Free VMware vSphere Hypervisor, Free Virtualization (ESXi)," *VMWare*, 03-Jul-2016. [Online]. Available: https://www.vmware.com/in/products/vsphere-hypervisor.html. [Accessed: 15-Dec-2016].

[2] "Network Virtualization and Security Platform – NSX," *VMWare*, 03-Mar-2016. [Online]. Available: https://www.vmware.com/in/products/nsx.html. [Accessed: 05-Jan-2017].

[3] "Server Virtualization with VMware vSphere," *VMWare*, 21-Jun-2016. [Online]. Available: https://www.vmware.com/in/products/vsphere.html. [Accessed: 15-Feb-2017].

[4] "Inventory Dataflow Diagram", Company Documentation of VMWare.

[5] Deepti Theng and K. N. Hande. "VM Management for Cross-Cloud Computing Environment." *Communication Systems and Network Technologies (CSNT)*, 2012 International Conference on , vol., no., pp.731,735, 11-13 May 2012.

[6] R. Fildes and C. Beard. "Forecasting systems for production and inventory control." *International Journal of Operations & Production Management* vol. 12.5, pp. 4-27, 1992.

[7] Y. Li, X. Xu, and F. Ye. "Research on supply chain inventory optimization and benefit coordination with controllable lead time." *Wireless Communications, Networking and Mobile Computing*, 2007.

[8] S. K. Majhi and S. K. Dhal. "Placement of Security Devices in Cloud Data Centre Network: Analysis and Implementation." *Procedia Computer Science*, vol. 78, pp. 33-39, 2016.

[9] S. Zhang and X. Chen. "Cloud computing research and development trend." *Future Networks, 2010. Second International Conference on* IEEE, 2010.

[10] P. Golle, J. Staddon and B. Waters. “Secure conjunctive keyword search over encrypted data.” ACNS, vol. 4, 2004.

[11] A. Gunasekaran, H. J. Williams and R. E. McGaughey. “Performance measurement and costing system in new enterprise.” *Technovation* vol. 25.5, pp. 523-533, 2005.

[12] C. Gong, J Liu, Q. Zhang and H. Chen. “The characteristics of cloud computing.” *Parallel Processing Workshops (ICPPW), 39th International Conference on* IEEE, 2010.

[13] H. Krawczyk, R. Canetti and M. Bellare. “HMAC: Keyed-hashing for message authentication.”, *RFC 2104*, 1997.

[14] S. L. Anil, and R. Thanka. “A survey on security of data outsourcing in cloud.” *International Journal of Scientific and Research Publications* on vol. 3, 2013.

[15] R. Fildes, and C. Beard. “Forecasting systems for production and inventory control.” *International Journal of Operations & Production Management* vol. 12.5 pp. 4-27, 1992.

[16] Tersine and Richard J. “Materials management and inventory systems” *Principles of inventory and materials management (2nd ed).* North Holland, New York, 1982.

[17] “Inventory Management System Protocol”, Company Documentation of VMWare.

[18] M. Mathur and A. Kesarwani. “Comparison between Des, 3des, Rc2, Rc6, Blowfish And Aes.” *Proceedings of National Conference on New Horizons in IT-NCNHIT*. vol. 3, 2013.

[19] Lanman, James, “Inventory management improvement techniques.” Masters Theses. Paper 1071, 2005.

[20] M. Alhanjouri, and A. M. Al Derawi. “A New Method of Query over Encrypted Data in Database using Hash Map.” *International Journal of*

Computer Applications vol. 41.4, 2012.

[21] Mrs.S.Selvarani, Dr.G.Sudha Sadhasivam. “Improved cost-based algorithm for task scheduling in Cloud computing.” *Computational Intelligence and Computing Research, international conference on* IEEE, 2010.

PUBLICATIONS

[1] Navjot Kaur and Ashima Singh. “Data Efficient NSX Inventory State Management”. *IEEE International Conference on Advanced Computational and Communication Paradigms – 2017 (ICACCP-2017).* [communicated]