UNIVERSITÉ DE MONTRÉAL

TOWARDS IMPROVING THE RELIABILITY OF LIVE MIGRATION OPERATIONS
IN OPENSTACK CLOUDS

ARMSTRONG TITA FOUNDJEM
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

TOWARDS IMPROVING THE RELIABILITY OF LIVE MIGRATION OPERATIONS
IN OPENSTACK CLOUDS

présenté par: FOUNDJEM Armstrong Tita
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. GUÉHÉNEUC Yann-Gaël, Doctorat, président
M. KHOMH Foutse, Ph. D., membre et directeur de recherche
M. QUINTERO Alejandro, Doctorat, membre

ACKNOWLEDGMENT

RÉSUMÉ

Grâce au succès de la virtualisation, les solutions infonuagiques sont aujourd'hui présentes dans plusieurs aspects de nos vies. La virtualisation permet d'abstraire les caractéristiques d'une machine physique sous forme d'instances de machines virtuelles. Les utilisateurs finaux peuvent alors consommer les ressources de ces machines virtuelles comme s'ils étaient sur une machine physique. De plus, les machines virtuelles en cours d'exécution peuvent être migrées d'un hôte source (généralement hébergé dans un centre de données) vers un autre hôte (hôte de destination, qui peut être hébergé dans un centre de données différent), sans perturber les services. Ce processus est appelé migration en temps réel de machine virtuelles. La migration en temps réel de machine virtuelles, est un outil puissant qui permet aux administrateurs de système infonuagiques d'équilibrer les charges dans un centre de données ou encore de déplacer des applications dans le but d'améliorer leurs performances et–ou leurs fiabilités. Toutefois, si elle n'est pas planifiée soigneusement, cette opération peut échouer. Ce qui peut entraîner une dégradation significative de la qualité de service des applications concernées et même parfois des interruptions de services. Il est donc extrêmement important d'équiper les administrateurs de systèmes infonuagiques d'outils leurs permettant d'évaluer et d'améliorer la performance des opérations de migration temps réel de machine virtuelles. Des efforts ont été réalisées par la communauté scientifique dans le but d'améliorer la fiabilité de ces opérations. Cependant, à cause de leur complexité et de la nature dynamique des environnements infonuagiques, plusieurs migrations en temps réel de machines virtuelles échouent encore.

Dans ce mémoire, nous nous appuyons sur les prédictions d'un modèle de classification (Random Forest) et sur des politiques générées par un processus de décision markovien (MDP), pour décider du moment propice pour une migration en temps réel de machine virtuelle, et de la destination qui assurerait un succès a l'opération. Nous réalisons des études de cas visant à évaluer l'efficacité de notre approche. Les défaillances sont simulées dans notre environnement d'exécution grâce à l'outils DestroyStack. Les résultats de ces études de cas montrent que notre approche permet de prédire les échecs de migration avec une précision de 95%. En identifiant le meilleur moment pour une migration en temps réel de machine virtuelle (grâce aux modèles MDP), en moyenne, nous sommes capable de réduire le temps de migration de 74% et la durée d'indisponibilité de la machine virtuelle de 21%.

ABSTRACT

Cloud computing has become commonplace with the help of virtualization as an enabling technology. Virtualization abstracts pools of compute resources and represents them as instances of virtual machines (VMs). End users can consume the resources of these VMs as if they were on a physical machine. Moreover, the running VMs can be migrated from one node (Source node; usually a data center) to another node (destination node; another datacenter) without disrupting services. A process known as live VM migration. Live migration is a powerful tool that system administrators can leverage to, for example, balance the loads in a data center or relocate an application to improve its performance and–or reliability. However, if not planned carefully, a live migration can fail, which can lead to service outage or significant performance degradation. Hence, it is utterly important to be able to assess and forecast the performance of live migration operations, before they are executed. The research community have proposed models and mechanisms to improve the reliability of live migration. Yet, because of the scale, complexity and the dynamic nature of cloud environments, live migration operations still fail.

In this thesis, we rely on predictions made by a Random Forest model and scheduling policies generated by a Markovian Decision Process (MDP), to decide on the migration time and destination node of a VM, during a live migration operation in OpenStack. We conduct a case study to assess the effectiveness of our approach, using the fault injection framework DestroyStack. Results show that our proposed approach can predict live migration failures with and accuracy of 95%. By identifying the best time for live migration with MDP models, in average, we can reduce the live migration time by 74% and the downtime by 21%.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABREVIATIONS

RISULM	Reinforced and Supervized Learning Model.
IT	Information Technology.
SLA	Service Level Agreement.
IaaS	Infrastructure as a Service.
SaaS	Software as a Service.
PaaS	Platform as a Service.
DC	Datacenter.
RL	Reinforced Learning.
ML	Machine Learning.
SL	Supervised Learning.
OS	Operating System.
OSS	Open Source Software.
VM(s)	Virtual Machine(s).
VMM	Virtual Machine Monitor.
MDP	Markov Decision Process.
KVM	Kernel-based Virtual Machine.
DT	Decision Tree.
DSS	Decision Support System.
RF	Random Forests.
NIC	Network Interface Card.
MLDO	Machine Learning based Downtime Optimization.
LAN	Local Area Network.
WAN	Wide Area Network.
LBFT	Load Balance Fault Tolerance.
ACC	Accuracy.
TPR	True Positive Rate.
TNR	True Negative Rate.
PPV	Positive Predictive Value.
NPV	Negative Predictive Value.

The results of this study have been submitted for publication :

- Towards Improving Live Migration's Reliability In OpenStack Cloud Armstrong Foundjem and Foutse Khomh, submitted for publication in *Proceedings of the 26th International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'17), Washington D.C., United States, June 26 - 30, 2017.*

My contribution:

Methodology, analysis, and paper writing.

CHAPTER 1   INTRODUCTION

Nowadays, cloud computing is attracting many organizations thanks to its economical benefits (Wang *et al.*, 2016). Both industrial and academic organizations are moving applications to the cloud (Jamshidi *et al.*, 2013) (Armbrust *et al.*, 2010). Users can lease the services of these applications, ramping up or down their capacity and paying only for what they use.

Applications deployed in the cloud typically run in virtual machines (VMs) or containers, distributed across multiple physical machines hosted in data centers. To respond to changing users' demands, organizations often have to migrate VMs out of overloaded and–or–overheated servers. VM migration is also necessary during servers' maintenance. When running instance(s) of a VMs are moved (migrated) from a source node, say Node A (or a datacenter) to a destination node, say Node B (or another datacenter), it is important to keep service downtime to its minimum.

Three VM migration techniques are currently supported in cloud environments: Cold migration (which consists in moving VMs that are not running from one Node to another), Live Block migration (consisting in moving live VMs across Nodes without copying shared memory location and VM disk over the network), and Live migration (consisting in moving live VMs across nodes with shared memory location being copied over the network). During Live migrations, the VMs memory state is copied from a source node to destination Nodes. In this work, we implements both the pre and post-copy approach of copying VMs memory states from source nodes to destination nodes. The pre-copy approach is fully supported in all releases of OpenStack and in most hypervisors. However, the post-copy approach is support only for releases beyond Mitaka. In order to have a larger impact, we experiment with both approaches in this study.

The pre-copy approach of VMs live migration consists in five steps: pre-migration, reservation, iterative pre-copy, stop and copy, and finally commitment. Figure 3.3 presents the details of these steps.

The post-copy approach also consists of five steps: pre-migration, reservation, stop & copy, post-copy: paging and finally commitment. Figure 3.4 provides a detailed description of each of these steps.

There are many benefits for doing live migration Clark *et al.* (2005); these include load balancing (work-load is distributed among computer nodes in an evenly manner to optimize the use of available computation power–CPU and resources), server or datacenter consolidation, and disaster recovering (VMs are evacuated from the disaster areas to safer areas).

However, live migration operations are not always successful. At times, failures may occur because

of issues with the state of the nodes, or even the state of the VM itself. These states can be described in terms of CPU, RAM (memory), or Network traffic (bandwidth) utilization etc., or a combination of these resources. For example, a VM that is running a high intensive memory or CPU application may fail live migration if the transfer of its state takes a longer time than it is allowed. Also, a higher traffic over the network during live migration can cause the VM migration process to fail.

1.1   Research Objectives

This work aims at studying failures of VMs during live migration (if the live migration will be successful or fail). Moreover, we want to make optimal decisions about ***where*** (the destination Node –datacenter to sent the running VMs to) and ***when*** (the appropriate time to live migrate) to live migrate VMs, with the aim to minimize the migration time T_m and the downtime T_d of the VMs. To achieve these goals, we have developed RISULM, a ReInforced and SUpervised Learning Model that allows organizations to predict the failure of VMs (if migrated), and identify the time of migration and the destination host that would ensure a minimum down time. We have implemented RISULM on OpenStack, using both the KVM and Xen hypervisors, with pre-copy and post-copy algorithms. We selected OpenStack because it is open-source, which allows us to modify its source code. Both KVM and Xen meets the required condition for OpenStack to implement pre-, and post-copy algorithm on live migration.

To assess the effectiveness of our proposed approach, we conduct a series of experiments comparing the effectiveness of RISULM with those of Machine Learning based Downtime Optimization – MLDO (Arif *et al.*, 2016), and load balance fault tolerance Strategy – LBFT   (Li et Wu, 2016) approaches. Specifically, we address the following two research questions :

RQ1 – Can we accurately predict live migration failures?
Result: We capture the state of the environment i.e., the CPU%, RAM%, Network-bandwidth utilization of the running virtual machines, the source Node and destination Node on both idle and loaded conditions. We use these parameters to train a Random Forest model. The training was performed in real time, as changes occurred in the environment. Results show that our proposed approach can predict live migration failures with 95% accuracy, using all combinations of algorithms and hypervisors, whereas, MLDO achieves a 92% accuracy with both pre-copy (KVM, Xen) and a 85% accuracy for post-copy (KVM, Xen). Also, LBFT achieves a 90% accuracy for pre-copy (KVM, Xen) and a 88% accuracy for post-copy (KVM, Xen).

RQ2 – Can live migration scheduling be improved using Markov Decision Processes?

Result: We implemented RISULM using Markov decision processes (MDP) to decide *when* and *where* to migrate VMs. We compared the performance of RISULM against the native OpenStack filter, MLDO, and LBFT approaches. Results show that RISULM can reduce VM migration failures by 23.3%, migration time by 74%, and VM downtime by 21%, whereas MLDO can reduce migration failure only by 14.6%, migration time by 63% and downtime by 13%. Also, LBFT can reduce migration failures only by 10.2%, migration time by 55% and downtime by 11%, using post-copy KVM combination.

1.2 Concepts and Definitions

This section provides brief summaries of concepts and definitions, that relate to our research work.

Cloud Computing: (Qian *et al.*, 2009), the origin of cloud computing can be traced back to the following prediction made in 1961 by John McCarthy : "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility. The computer utility could become the basis of a new and important industry."
Earl et al. (Erl *et al.*, 2013) define cloud computing as "A specialized form of distributed computing, which introduces utilization models for remotely provisioning scalable and measured resources."

On-premises: IT resources (utilities) that are locally available on the hosting enterprise (premises). It is mutually exclusive to cloud computing resources (utilities).

Public Cloud: Cloud infrastructure whereby resources or utilities are made publicly available over the Internet.

Private Cloud: Cloud infrastructure whereby resources or utilities are made privately available only to particular organizations and are not made available to the entire Internet.

Community Cloud: Cloud infrastructure whereby resources or utilities are made available to a group of organizations that form a community with common concerns.

Hybrid Cloud: A combination of both private and public clouds. Here organizations choose what to share over the Internet and what to make private.

Vertical Scaling: Increasing or decreasing the capacity of an existing commodity such as hardware or software, by adding or reducing its computational power.

Horizontal Scaling: Adding or removing hardware nodes in a cluster to increase or reduce its capacity.

Open Source Software: The source code of a computer program that is made publicly available with a license that allow users to access, study, and enhance. Whereas, enhanced works might not be distributed freely.

Node: A physical computer also referred to as Host, in which VMs are managed i.e., created, hosted, etc.

Latency: Network latency refers to the time delay data packet takes to go from one Node to another.

datacenter: A computer facility center where pools of storage, network, compute and associated components are hosted.

Kernel-based Virtual Machine (KVM): A full virtualization infrastructure that turns the Linux kernel into an hypervisor.

IaaS: Infrastructure as a Service, provides users with virtualized computing utilities, such as an entire datacenter over the Internet, or a pool of VMs. Openstack clouds are examples of IaaS.

PaaS: Platform as a Service enables users to deploy or manage their applications. It hides the complexity of the underlying virtualized severs from users. An example of PaaS is Microsoft Azure.

SaaS: Software as a Service enables users to have access to all the functionalities of a conventional software applications through a web portal. Users can use this software on a pay-as-yo-go bases. An example of SaaS is GMAIL.

Hypervisor: A hypervisor also known as Virtual Machine Monitor (VMM) is a piece of hardware, firmware, or a computer software, that enables virtual machine management such as creating, running, live migrating, and deleting virtual machines.

Virtualization: Virtualization is the abstraction of pools of compute resources. The abstracted resources are represented as instances of virtual machines (VMs) or containers. Virtualization is a key technology of cloud computing. A key benefit of Virtualization is the isolation; each VM is isolated from the physical system of the host and from other virtualized machines. Different types of server virtualization can be implemented in a cloud environment:

- *Full Virtualization* virtual machines running on the guest OS are completely isolated and simulates the entire underlying hardware;

- *Partial Virtualization* allows a portion of the underlying hardware to be simulated. Consequentially, some guest applications might required modifications so as to function properly in the virtualized environments;

- *Paravirtualizaiton* the guest OS is recompiled before it is installed inside a virtualized environment, in order to achieve the enhancement of virtualization. Whenever the guest OS issues a call to the hardware, it is substituted instead with calls to the VMM;

- *Hardware-assisted virtualization* a platform virtualization, which provides full virtualization with the help of the underlying hardware capabilities, and finally,

- *OS-level Virtualization* no hypervisor is needed for this type of virtualization. The host OS has virtualization capabilities that allow him to play the role of an hypervisor.

1.3   Thesis Plan

The rest of this thesis is organized as follows. In Chapter 2 we review the related literature and in Chapter 3 we give a general overview of OpenStack and live migration in OpenStack cloud. In chapter 5, we describe the Methodology used in this study and explain how we implemented RISULM. In Chapter 6 we discuss the results of our case study that aimed at evaluating the effectiveness of our proposed solution. In chapter 7 we present a summary of our work and outlines some avenues for future works.

CHAPTER 2   LITERATURE REVIEW

Several works on live migration, using different types of hypervisors (such as KVM, QEMU, Xen etc.) exist in the literature. Some of these works have proposed models to predict the performance of live migration operations, and mechanisms to optimize migration time. In this section, we discuss works that are closely related to our research direction.

2.1   Live migration in Cloud Environment

(Biswas *et al*., 2016) investigated performance issues caused by improper resource management, during live migration in Openstack. To conduct their study, the authors implemented a transatlantic optic fiber network, and performed post-copy live migration of VMs, using KVM in OpenStack. Memory intensive applications were running in the migrated VMs during the experiment. Through these experiments, they observed that specific VM memory size patterns, under loaded or idle conditions, can influence both the total migration time and the network data transfer rate. They also observed that the network distance has no significant effect on the downtime of the migrated VM. Similar to this study, our proposed approach RISULM uses VM size, CPU, Network, and workload characteristics to predict failures prior to the migration of VMs. However, during our experimentation, we considered both CPU and memory intensive applications, while this previous work considered only memory intensive applications. Also, we are able to predict where and when to migrate the VMs.

(Bunyakitanon et Peng, 2014) investigated the possibility of automating the live migration of VMs in a cloud environment. A mechanisms that can significantly improve the load balancing of VMs in a datacenter. The authors implemented Push and Pull Hybrid strategies, capable of identifying heavy loaded and less-loaded Nodes in a datacenter, based on the characteristics of the Nodes. They claim that using these strategies, system administrators will be able to automatically decide on which VM to live migrate, and where to migrate the VM (the destination Node).

To evaluate their proposed strategies, the authors performed several migration attempts and measured the response time of the migrated VMs. They report that the response time of migrated VMs is not affected by the workload being executed in the VM. A result that is not confirmed by our study. In fact, we observed that workload characteristics can help predict VM live migration failures. Nevertheless, they were able to validate their findings both through simulation and on a testbed of workloads running in VMs created by both KVM and Xen. The VMs created with Xen had better response times, while the number of migration attempts was lower for VMs created with

KVM.

(Gustafsson, 2013) proposes a technique to reduce the total migration time of VMs, during live migration operations.

The proposed techniques is based on a strategy that consists in sending only a subset of data that are critical over the network. The remainder of the data is available through the shared storage location.To assess the effectiveness of their proposed approach, the author implemented several Linux benchmarks and performed VM migrations in pre-copy mode. The migrated VMs were created using Xen.

One average, their proposed approach could reduce the total migration time by 40.48 seconds, which was a 44% reduction of the total migration time. This approach is however limited to the Xen hypervisor and implements only the pre-copy algorithm, whereas, our approach (i.e., RISULM) implements both the pre- and post-copy algorithms on Xen and KVM hypervisors. Moreover, results show that RISULM can achieve a better performance (in terms of total migration time) on Xen, than their proposed approach.

(Liaqat *et al.*, 2016) discusses issues related to memory transfer during live migration.

They propose to assign adaptive priorities to pages during VM live migrations, based on system, application, and service demands. They categorized VM memory pages into several classes (depending on the type of the pages) and assigned different priorities to the classes. They also suggested the use of scheduling algorithms, such as round robin, to assign time-slots to each page to transfer. However, this work is mostly conceptual and no concrete implementation has been provided yet.

2.2   Performance Modeling of Concurrent Live Migration

(Kikuchi et Matsumoto, 2011) addresses the problem of multi-tenant concurrent live migration of VMs in the cloud, which may result in spikes and–or outbursts that can degrade the performance of VMs.

The authors propose a performance model, which evaluates parallel operations of VMs live migration in datacenters. This model uses PRISM ("a probabilistic model checker"), which enables the performance model to derive the verification results as probability values and not as parameter values. This probability values makes it easier for cloud system administrators to determine their level of confidence before conducting live migration operations. Moreover, because the verification results are obtained from a model checker, it is not necessary to re-run the experiments several times, to obtain the probabilistic results for a given level of confidence.

The authors were able verify the PRISM language properties regarding the performance of live

migration using their proposed model.

Results suggest that VMs have high chances to hang during live migration when multiple sender servers target a single receiver, due to the delays incurred by the processing of concurrent live migrations at the receiver side.In the future, we plan to verify RISULM using formal verification techniques.

2.3   Prediction on Live Migration

(Arif *et al*., 2016) investigated live migration operations in Wide Area Networks (WAN) and observed that, although several approaches have been proposed to reduce downtime during live migration, none have considered the case of VM migrations through a WAN.
To help fill this gap in the literature, they designed MLDO; a Machine Learning based Downtime Optimization approach that uses predictive mechanisms to estimate the downtime of VMs during live migration operations over WAN networks. The input of MLDO are system's parameters (e.g., CPU, RAM, Network, and the load conditions of the machines). Our approach RISULM also uses these parameters. MLDO has two complementary modules: a *monitoring* and a *processing* module. Similarly to MLDO, RISULM also rely on two complementary models to make its decisions.

The monitoring module of MLDO collects data about changes occurring in the system (both in physical and virtual machines). These data are used by the processing module to train classifiers (e.g., the c4.5 algorithm) that are used to generate VM migration decisions. Using MLDO, the authors were able to reduce downtime by 15%, during VM migration. In this thesis, in addition to minimizing VM downtime, we also aim to reduce the migration time. In our analysis, we implement and compare the performance of RISULM with that of MLDO.

 (Li et Wu, 2016) investigated load balancing mechanisms and proposed LBFT; a Load-Balancing and Fault Tolerance Strategy that can help stakeholders decide about when to migrate a VM. Their proposed strategy consists in monitoring resource usages, especially the patterns at which memory is accessed by the workloads running on the nodes; in order to derive threshold values that can help identify busy nodes that should be migrated to balance the workload in the cloud environment. Similarly to LBFT, our approach also monitors the system to know which nodes are less busy, and hence could receive more VMs.

The authors conducted a series of experiments to assess the effectiveness of their proposed approach, and concluded that LBFT can accurately predict migration time in 90% of cases, saving between 35% to 50% of migration cost, in comparison to a random strategy. In this thesis, we compare the performance of RISULM against the performance of LBFT.

(Nathan *et al.*, 2015) investigate factors that affect live migration in cloud environments, and which they claim have been neglected by prior works.

They propose a model to predict VMs live migration time that uses both KVM and Xen hypervisors, and the following three parameters: the size of the working set that is writable, the amount of pages that the skip technique can be applied to, the correlation between the amount of skipped pages with respect to page dirty rate, and the rate at which pages are transferred. The built models containing different combinations of the aforementioned parameters and compare their performance on both KVM and Xen. The results obtained on 53 workloads show that on average, their proposed models can predict VM migration times with an error of 43 secs for KVM and 112 secs for Xen. Their results also show that KVM performs better than Xen in terms of live migration time. Our work goes further beyond this previous work, and use both pre-copy and post-copy algorithms on live migration of VMs created using KVM and Xen. We also report better results, in terms of migration time and downtime reduction.

(Akoush *et al.*, 2010) investigated parameters that affects live migration time of VMs and proposed two models that system administrators can use to predict VMs migration time: HIST (History Based Page Dirty Rate), and AVG (average page dirty rate). For the AVG model, page dirty rates are assumed to be constant for all the VMs. The proposed models were tested under several conditions, and results show that, in certain conditions, the AVG model can become impractical. Results also show that both AVG and HIST can predict the total migration time of VMs with a 90% accuracy, both on synthesized and real-world workloads. The proposed models (HIST, AVG) are tied to Xen because they use built-in functions of Xen. To over come this limitation, our proposed model RISULM uses different types of hypervisors and both pre-, and post-copy algorithms.

(Barrett *et al.*, 2011) investigated scheduling issues in scientific workflow problems, which often require extensive computational power, and generate significant data during experiments. Efficient schedulers are required for workflow based applications, because the multiple dependencies that exist among the tasks make their planning difficult. For example, all children tasks must only start when their parents are completed.

The authors proposed a novel cloud workflow scheduling system, which is composed of three main parts: Scheduling Engine (consisting of the genetic algorithm – solver Agents), Work-flow Management System (consisting of a user interface, a planner and an Executor) and the Cloud (uses GoGrid on Amazon EC2).

This system uses a Markov Decision Process (MDP) in the workflow execution process. The MDP iteratively monitors the cloud base system (GoGrid) to make decisions. Our approach also uses an

MDP to decide about when and where to live migrate VMs.

To evaluate their proposed scheduling system, the authors used Cloudsim, which is a cloud based simulator. Results suggest that their proposed system can choose optimally from a set of workflow schedules, even though, it initially incur high cost because it assigns the same probability to every actions. Which results in time being wasted exploring less optimal actions, which are penalized.

Our RISULM implementation addresses this limitation by ensuring that no extreme probability value (e.g., 0 or 1) is assigned to a node. For each node, we have a possibility of the VM not to migrate, which implies that it can stay in the same node with a certain probability.

CHAPTER 3   OPENSTACK CLOUD

3.1   OpenStack Overview

OpenStack is an open source operating system for cloud computing, which was initiated by the National Aeronautics and Space Administration (NASA). In 2010, Rackspace Cloud joined the project and together they created the first release of OpenStack. Since then, OpenStack has grown into a mature project, supported by the OpenStack Foundation, with over 1M lines-of-code and over 94K commits mostly writing in python. OpenStack is freely available under Apache license 2.0. Releases of OpenStack are named using alphabetical order from Austin "A" to Newton "N" etc. Most Linux distributions support OpenStack, which is why most end users prefer OpenStack, since they can easily install and manage the platform on their preferred Linux distribution that they are already familiar with.

In July 2011, OpenStack was adopted by Ubuntu developers. Consequently, Ubuntu Linux became the default reference implementation of OpenStack, with rich sets of documentations than any other Linux distribution. That is why in our study, we are using OpenStack on Ubuntu.

OpenStack clouds provide Infrastructure-as-a-Service (IaaS) to users (see Figure3.1). These IaaS are composed of pools of compute, storage, and networking resources. Recently, the Technical Committee of the OpenStack Foundation (OSF), suggested changes in the mission statement that reads:

"*To produce the ubiquitous Open Source Cloud Computing platform that enables building interoperability public and private clouds regardless of size, by being simple to implement and massively scalable while serving the cloud users' needs.*" [1]

OpenStack aims at implementing scalable and rich sets of features; this is achieved through the contributions of dedicated cloud computing experts and software developers from around the world. OpenStack IaaS offer a variety of services. Each service can be accessed through an application programming interface (API) that facilitates its integration. In Table 3.1, we present the main services, project name and description that forms an OpenStack IaaS as defined by the OpenStack Foundation and in Figure 3.2 we present the logical architectural design of OpenStack[2].

---

[1]"https://governance.openstack.org/resolutions/20160106-mission-amendment.html"
[2]http://docs.openstack.org/admin-guide/common/get-started-logical-architecture.html

Among the available open source cloud IaaS that exist, such as: OpenStack, OpenNebula, Eucalyptus, Apache CloudStack, etc., OpenStack happens to be the IaaS with the widest coverage in terms of multiple services (projects) covering the majority of areas in cloud computing. It has a distributed architecture and it is mostly used by large commercial enterprises. The source code of OpenStack is mostly written in one language; Python, which are controlled/managed through APIs (Ismaeel *et al.*, 2015). OpenStack supports multiple hypervisors suck as KVM, Xen, QEMU, VMware, Hyper-V, etc., and containers (Bell *et al.*, 2013)(OpenStack, 2016).

As part of our study objectives, we are interested in comparing the performance of hypervisors that uses both the pre- and post-copy algorithms for live migration. We are also interested in having a scalable cloud infrastructure that allows us to add and remove services, and to modify the source code (in order to implement our proposed approach). For all these reasons, OpenStack is the best candidate for our study.
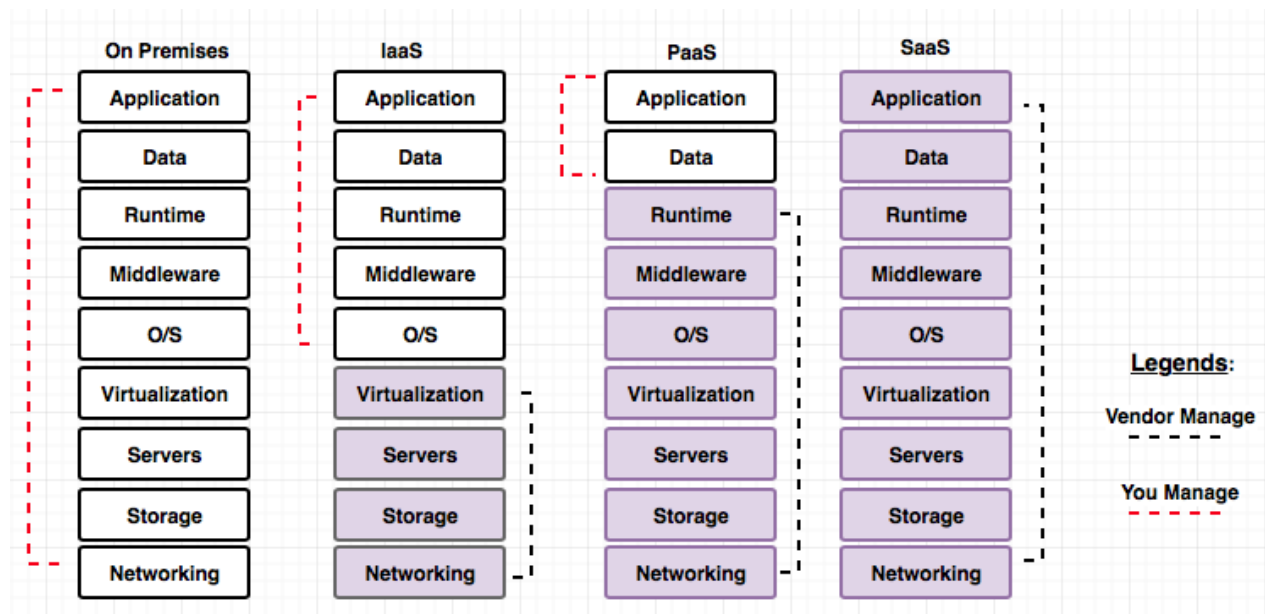
Figure 3.1 On-premises vs. Cloud Service Models Showing Separation of Responsibilities

Table 3.1 OpenStack services and projects described

Service	Project Name	Description
Dashboard	**Horizon**	Provides a web portal that facilitates interactions with other services in the stack. Users can create, lunch, or migrate VM instances.
Compute	**Nova**	The principal service of the stack that manages pools of compute resources such as instances in the IaaS stack environment. It is responsible for scheduling, spawning, provisioning resources and terminating virtual machines on demand.
Networking	**Neutron**	The network service of the IaaS. It is responsible for providing Network-as-a-Service to other services in the stack. Users can define and customize their network with neutron using its API.
Objects Storage	**Swift**	Provides storage and retrieval of unstructured objects in the stack, using a RESTful API. It supports replication and is highly scalable.
Block	**Storage**	Enables instances to use block storage that are persistent in the stack. It has a pluggable architecture that manages block storage devices in the stack.
Identity service	**Keystone**	Enables authentication and authorization as a service for other services in the stack.
Image service	**Glance**	Provides image storage and retrieval to other services in the stack. The Compute service uses the Image service when provisioning an instance.
Telemetry	**Ceilometer**	Enables the stack to monitor usages and activities of other services. It is principally use for billing but also provides benchmarking and statistics about the performance of the stack .
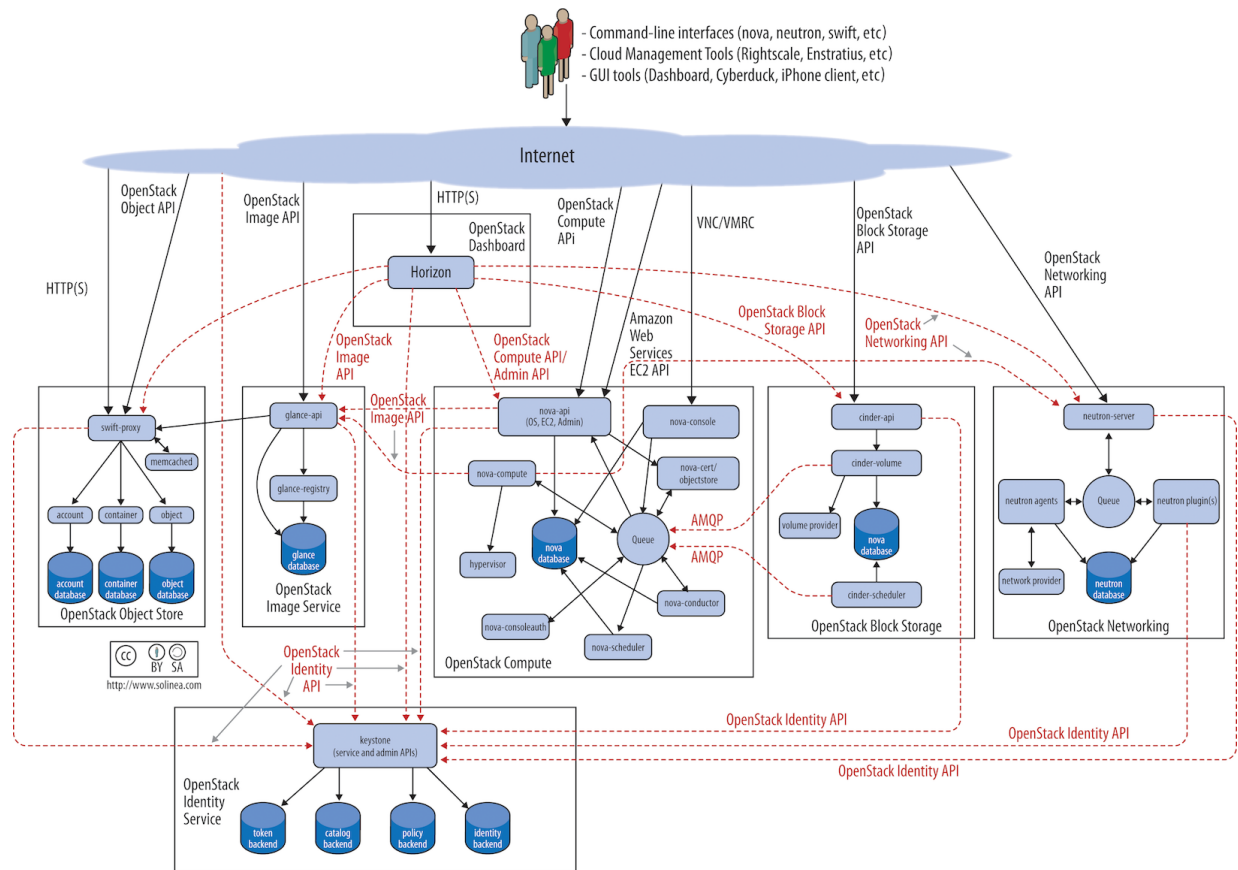
Figure 3.2 The OpenStack Logical Architecture, courtesy of the OpenStack Foundation.[3]

3.2   Live Migration in OpenStack Cloud

OpenStack fully supports live migration. In this section, we briefly explain how live migration is achieved in OpenStack cloud environments using both the pre- and post-copy algorithm as implemented in KVM and Xen hypervisors. First and foremost, for live migration to work, one needs a minimum of two physical machines. One of these physical machines should play the role of controller and there should be at least one compute node in the cluster. Moreover, all the physical machines that form the cluster (the computes and controller Nodes) should have identical CPUs, for compatibility purpose.

OpenStack live migration supports pre-copy live migration for all releases and post-copy live migration for releases starting from Mitaka and beyond. In our study we use both the pre-copy and post-copy, for this reason, we chose the Mitaka release of OpenStack that supports both pre- and post-copy algorithm. We setup Openstack on a cluster of identical machines.

---

[3]Source:"https://docs.openstack.org/arch-design/design.html"

Figure 3.3 and Figure 3.4 present the steps of the pre-copy and post-copy algorithms that are executed during a live migration of VM in OpenStack.

In Figure 3.5 we also show the life-cycle of VMs during the migration, i.e., the possible states of the VMs. Later on in this work, we will explain how we used these states to compute the migration and downtime of a VM.

OpenStack supports a variety of hypervisors as shown in Figure 3.6. However, not all hypervisors supports both the pre- and post-copy algorithm for live migration. Therefore, in this study, since we want to experiment on hypervisors that supports both the pre- and post-copy algorithm on live migration, we select the following hypervisors: **KVM** - Kernel-based Virtual Machine; and **Xen** (using libvirt) for paravirtualization.
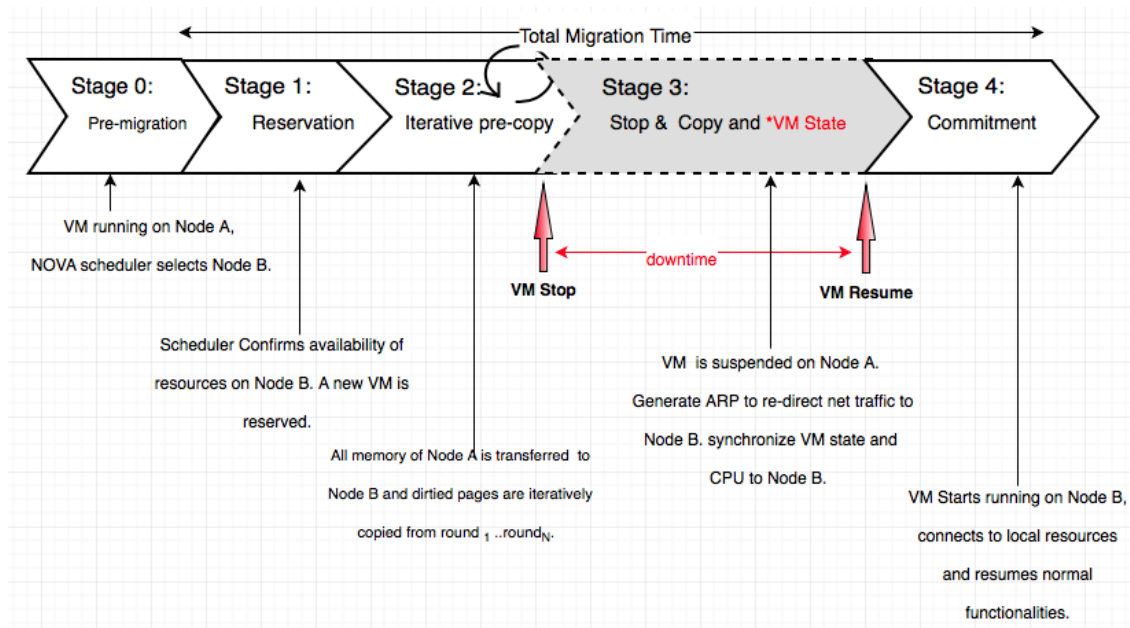
Figure 3.3 The live migration pre-copy algorithm.
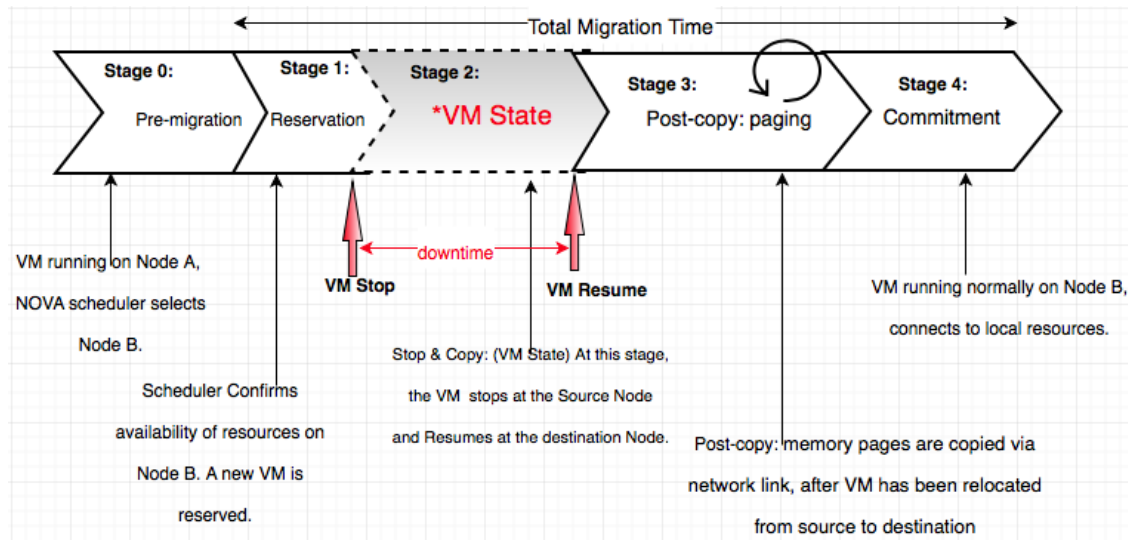
Figure 3.4 The live migration post-copy algorithm.
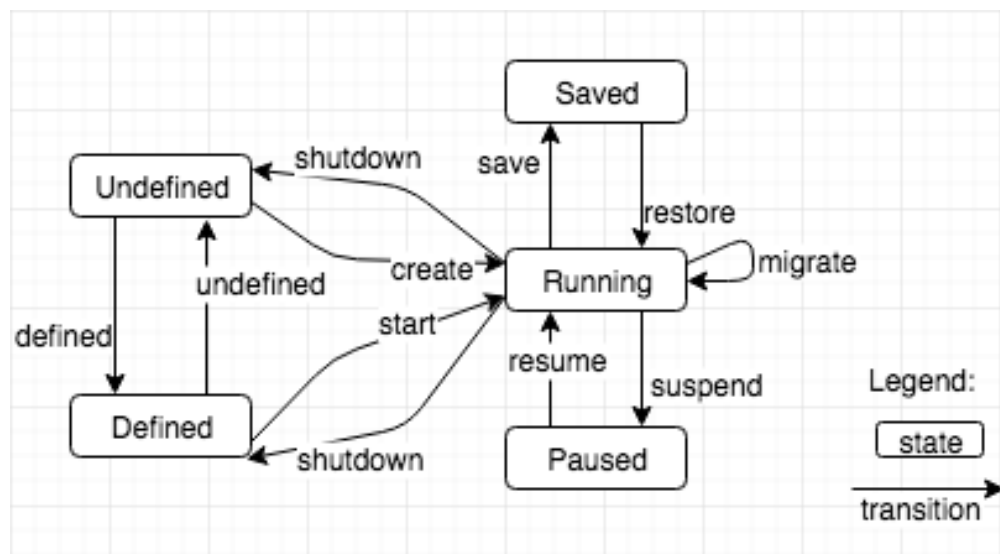
Figure 3.5 The possible states of a VM during a Live Migration

Feature	Status	Hyper-V	Ironic	Libvirt KVM (ppc64)	Libvirt KVM (s390x)	Libvirt KVM (x86)	Libvirt LXC	Libvirt QEMU (x86)	Libvirt Virtuozzo CT	Libvirt Virtuozzo VM	Libvirt Xen	VMware vCenter	XenServer
Attach block volume to instance	optional	✔	✗	✔	✔	✔	✗	✔	✗	✔	✔	✔	✔
Detach block volume from instance	optional	✔	✗	✔	✔	✔	✗	✔	✗	✔	✔	✔	✔
Set the host in a maintenance mode	optional	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✔
Evacuate instances from a host	optional	?	?	?	✔	✔	?	?	✗	✗	?	?	?
Rebuild instance	optional	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Guest instance status	mandatory	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Guest host status	optional	✔	✗	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Live migrate instance across hosts	optional	✔	✗	✔	✔	✔	✗	✔	✗	✗	✔	✗	✔
Force live migration to complete	optional	✗	✗	✔	✔	✔	✗	✔	✗	✗	✗	✗	✗
Launch instance	mandatory	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Stop instance CPUs (pause)	optional	✔	✗	✔	✔	✔	✔	✔	✗	✔	✔	✗	✔
Reboot instance	optional	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Rescue instance	optional	✔	✗	✔	✔	✔	✗	✔	✔	✔	✔	✔	✔

Figure 3.6 Matrix – Cross-section of Hypervisor supported by OpenStack Nova Live Migration.

CHAPTER 4   LEARNING MODELS

This chapter provides background information about the models used in this thesis.

4.1   Supervised and Reinforcement Learning Models

In this thesis, we propose a novel model (i.e., RISULM), which is made up of both a supervised model (random forests) and a Reinforcement learning model (Markov decision process). In this section, we will briefly explain how these models work and how we implemented them.
Supervised learning models regroup two main categories of algorithms: classification and regression algorithms. classification algorithms uses dataset that have categorical outcome values, that is, where the data can be partitioned into specific classes. On the other hand, regression algorithms uses dataset that have continuous outcome (output) value. The outcome of our dataset is either "Success" or "Fail", which is why we use a classification algorithm (i.e., Random Forests) instead of a regression algorithm.

4.1.1   Classification Algorithm

To answer our first research question, i.e., *RQ1: – Can live migration failures be accurately predicted?*, we selected the following machine learning techniques : Naive Bayes (NB), K - Nearest Neighbor (KNN with Euclidean distance), and Random Forests (RF). We choose these techniques because they have been used in previous studies from the literature to build predictive models (Caruana et Niculescu-Mizil, 2006), (Blaser et Fryzlewicz, 2016), (De Poalo et Howard, 2014), (Xiong *et al.*, 2012). We compared the performance of these algorithms and observed that RF consistently outperforms NB and KNN on our data set. Hence, we selected RF to be included in RISULM. In the following, we provide a brief description of RF models.

Random Forest Models

Random Forests (RF) uses a large number of classification trees, also known as decision trees (DTs) to make its classification decisions. These DTs are nourished by the observations and the most likely outcome from the DTs's observation is then passed as an outcome. When there is a new observation, it is fed to the trees that makes up the forest, each tree votes for a class (classification). The forest chooses the classification with the majority of votes among all the trees that forms the forest.

Each tree that builds up into a forests grows as follows: for a training set of size N, where N is a subset of the population P, (we use 80% : 20%, N=80%), the training set is sampled at random with replacement. The tree is then grow with the sample of N. For M input variables there exist m such that $m \ll M$, at each node, with the m variables randomly selected from M. The optimum split for m is the same that is used to split the node. Moreover, as the forest grows, m is held constant and each tree grows to its possible limit without pruning. Additionally, if correlation exist between any two trees, this will increase the error rate of the forest as it grow.

We implemented the k-fold cross-validation within the random forest, for k=10. The basic principle of cross validation is explained below.

- Split the dataset P into k-folds randomly, (our case, k=10)

- With each of the k folds in P, build a model using (k - 1 = 9) folds of P. Then, use the kth fold to test the model.

- keep track of each error as you do predictions.

- The previous steps are repeated until all k-fold are used as test set.

- The performance metric of the model is determined by the average error of k, which is known as cross-validation error.

The algorithm of our Rf model presented on Algorithm 1, takes the dataset and splits it into two (80 : 20), keeps track of the current state of resource utilization while the RF classifier builds the model until the stopping conditions are met. Since the models keeps running as long as nodes have VMs ready for live migration, the current utilization line 25 will always be above THD, therefore, the loops continue, however, if all Nodes are idle and the THD value becomes greater than the current utilization, the RF model stops. The outcome of our Rf model is Boolean (Yes or NO). The algorithm is summarized in Figure 4.1, were we show an example of how the dataset is split (training set over testing set), implementing the five steps of our algorithm on the training set and after, computing the confusion metrics, accuracy and prediction. We use the Python package (Buitinck *et al.*, 2013) to build our RF model.

Algorithm 1: Supervised_Learning(dataset)

Result: Migrate VMs at the optimum time (when)

Input   :

1 Extract the environmental variables that describe the state of the systems

Output   :

2 migration time, downtime

Local variable:

3 $CPU \leftarrow$ sum of *all current cpu utilization*

4 $RAM \leftarrow$ sum of *all current RAM utilization*

5 $BW \leftarrow$ sum of *all current network utilization*

6 $VMz \leftarrow$ size of *VMs to be migrated*

7 $bestTime \leftarrow t_0$

8 $currentTime \leftarrow$ track_current_system_time

9 $THD \leftarrow minVal$

10 $outcome \leftarrow null$

11 $current\_utilization \leftarrow med(RAM, CPU, BW, VMz)$

12 **repeat**

13     $trainSet, testSet \leftarrow split\_8020(datasetP)$

14     $model \leftarrow splitKfoldRandom(trainSet)$

15     $performMetrics \leftarrow (crossValidErr)$

16     $confusMatrics(testSet)$

17     $acc, outOfBag \leftarrow compute(confusMatrics)$

18   **while** *model* **do**

19   *nn*

20   **for** *values of RAM, CPU, BW, VMz* **do**

21             $newFeatures \leftarrow rf(RAM, CPU, BW, VMz)$

22             $predictModel \leftarrow (newFeatures, acc, outOfBag)$

23             $outcome \leftarrow predictModel$

24   **if** *outcome* **then**

                                            /* Migrate VMs here */

25   *migrate VMs*

26                 $bestTime \leftarrow currentTime$

27   **else**

28   continue

29   **end**

30   **end**

31   **end**

32 **until** *current_utilization < THD*

33 **return** *bestTime*

```
                              ┌─────────┐
                              │  Start  │
                              └─────────┘
                                   │
                                   ▼
              ╱─────────────────────────────────────────╲
              ╲  Data extraction from system; P=500      ╱
              └────────────────────┬────────────────────┘
                                   │
                                   ▼
        ┌──────────────────────────────────┐   Repead 10 times with different seed s
        │ Split data into Train and Test   │◄──────────────────────
        │ sets for 10-fold cross-validation.│
        └──────────────────────────────────┘
```

Model Selection and parameter Optimization steps:
1. Split the dataset P into k-folds randomly, (our case, K=10)
2. With each of the k folds in P, build a model using (k - 1 = 9) folds of P. Then, use the kth fold to test the model.
3. keep track of the each error as you do predictions.
4. The previous steps are repeated until all k-fold are used as test set.
5. The performance metric of the model is determined by the average error of k, which is known as cross-validation error.

Train Set; T=400 (80% of P)

Test set; S=100 (20% of P)

Compute the confusion matrix for S

Compute Accuracy and Out-Of-Bag

Predictive model

Real time data

New features

Repeat prediction until no input data

Predict outcome

No input data

End
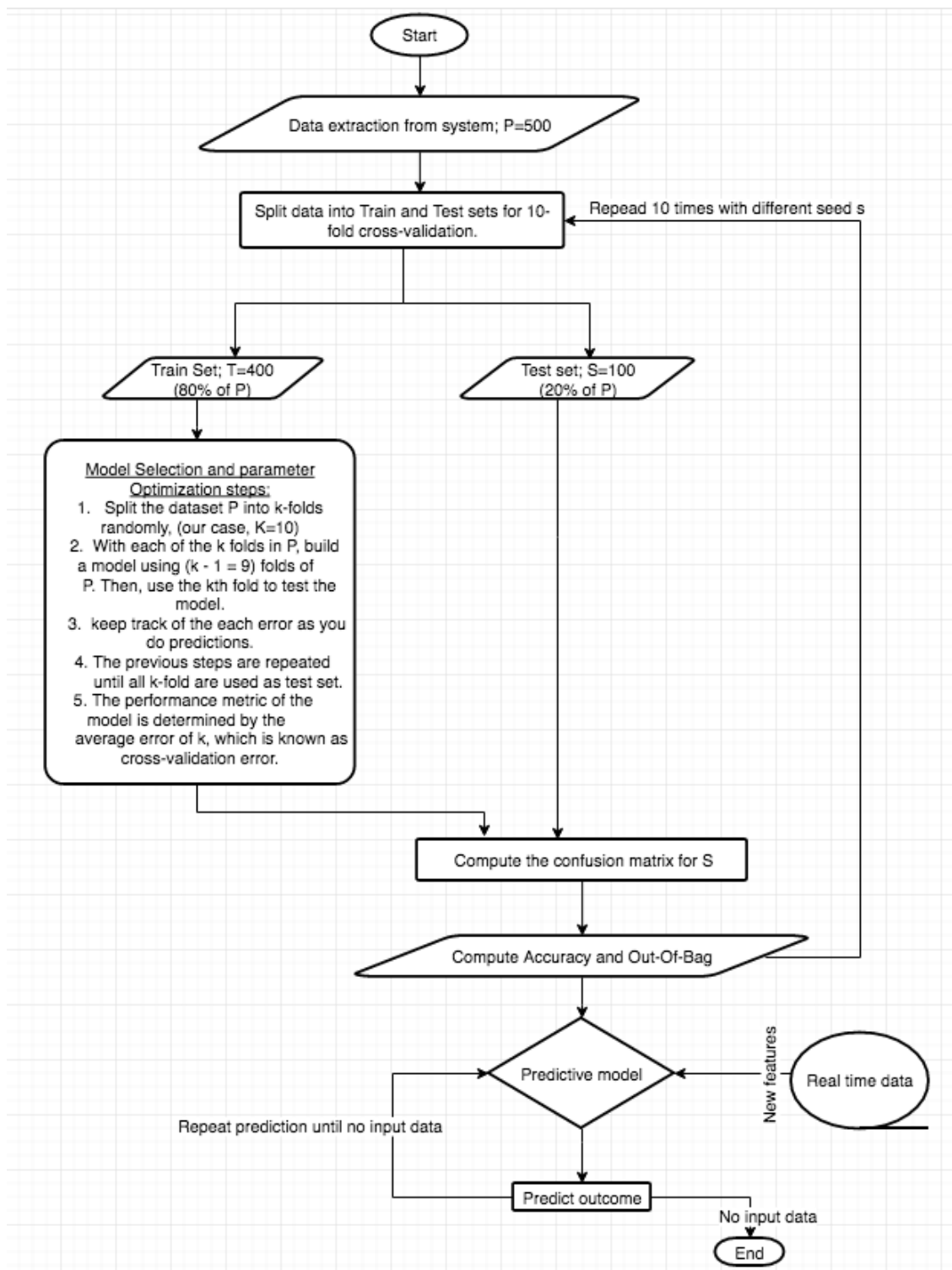
Figure 4.1 Supervised Learning Model

4.1.2   Reinforcement Learning : Markov Decision Process (MDP)

To answer our second research question, i.e., *RQ2 – Can live migration scheduling be improved using Markov Decision Processes?*, we implemented a Reinforcement learning scheduler based on Markov decision processes (MDP). This MDP technique uses the current state of the system to make decisions. It learns over time as the state of the system changes, for example in Figure 4.2, we show an MDP with five nodes (Nodes). These nodes represent our system. Suppose that one wants to migrate a VM that is hosted in N_1, the MDP will have to learn the optimal (best) policy first. That is, each time it takes an action, it assigns a probability and reward to it. If the migration fails the VM is rolled back to the node and we represent this with a loop back to the node with a low probability (0.1), since we don't encourages these types of actions, but can't avoid it as well. On the other hand, if the migration succeeds, we assign a probability and a reward to that action, depending on if we encourage this type of action or penalizes it. In some cases, we had equal probability but rewarded them differently, based on the current observed conditions of the system. If the VM goes to a node that is more busy or having more loads that the other nodes, we penalize this action. For example, N_1 to N_4 has a negative reward but equal provability with N_1, N_3 and N_5. After policy iteration, N_4 will be avoided and N_2 preferred based on that specific state of the system. The MDP hence learn the best path, even though the system changes over time. In subsequent runs, the MDP will reads the current state of the system and make timely decision without re-learning the path over and over.

An MDP is defined as a tuple $\langle \mathscr{S}, \mathscr{A}, \mathscr{P}, \mathscr{R}, \gamma \rangle$ where,

- $\mathscr{S}$ represents our set of finite states,

- $\mathscr{A}$ represents our set of finite actions,

- $\mathscr{P}$ represents a transitional probability function moving from state *s* to state *s'* when action *a* is taken,

- $\mathscr{R}$ represents an immediate reward that is obtained when action *a* is taken,

- γ represents a discounting, which can be finite or infinite. This actually determines our termination criteria.

- π represents a policy that maps a state *s* to an action *a*. Our goal using MDP is to find optimal policy that maximizes reward, as we iterate through the policies.

Our implementation of MDP uses policy iteration as shown in Algorithm 2. The purpose here is to return the optimal policy, which is the best time to migrate a VM and the available destination.

The process continues until there are no more changes in the system state, line 11 to 26. The most subtle part of policy iteration is line 13, we will not go into the detail here. We loop from line 14, 16 to 24 and 25 observing if there is a change of policy, in the end, we return the optimal policy at line 28. The simplified flowchart of our MDP is shown in Figure 4.3.
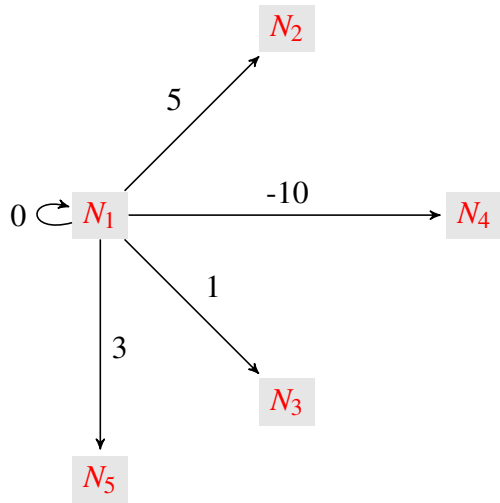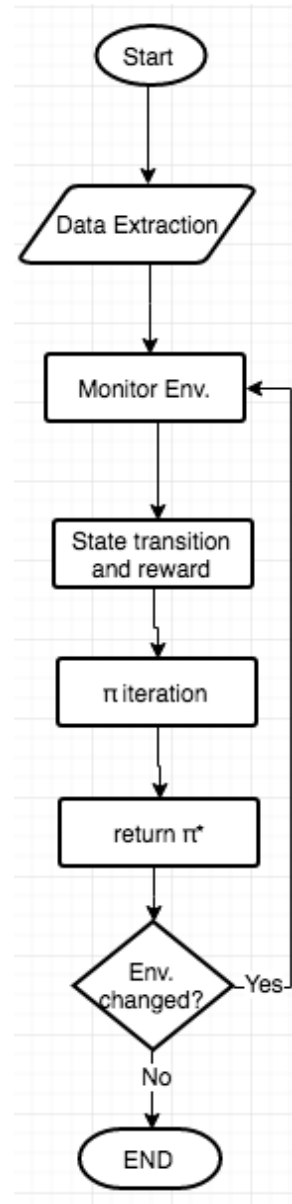
Figure 4.2 MDP use-case live migration.

Figure 4.3 Flowchart of our MDP approach.

Algorithm 2: Policy_Iteration($\mathscr{S}, \mathscr{A}, \mathscr{P}, \mathscr{R}, \gamma$)

   Input   :

1 Extract the environmental variables that describe the current state of the systems

2 $\mathscr{S} \leftarrow \{s_1, s_2, \ldots, s_n\}$

3 $\mathscr{A} \leftarrow \{a_1, a_2, \ldots, a_n\}$

4 $\mathscr{P} \leftarrow P(s'|s, a)$

5 $\mathscr{R} \leftarrow R(s, a, s')$

6 $\gamma \in \,]-1, 0[$

   Output   :

7 optimal policy π^*

   Local variable:

8 action_array $\pi[S]$

9 real_array $\mathscr{V}[S]$

10 $\pi^* \leftarrow \pi'$

11 **repeat**

12     $sysChange \leftarrow False$

13     $\mathscr{V}[S] = \sum_{s' \in S} P(s'|s, \pi[s])(R(s, a, s') + \gamma \mathscr{V}[s'])$

14   **for** *each* $s \in \mathscr{S}$ **do**

15         $\mathscr{Q}\_global \leftarrow \mathscr{V}[s]$

16   **for** *each* $a \in \mathscr{A}$ **do**

17             $\mathscr{Q}_{s,a} = \sum_{s' \in S} P(s'|s, a)(R(s, a, s') + \gamma \mathscr{V}[s'])$

18   **if** $\mathscr{Q}_{s,a} > \mathscr{Q}_{global}$ **then**

19                 $\pi[s] \leftarrow a$

20                 $\mathscr{Q}_{global} \leftarrow \mathscr{Q}_{s,a}$

21                 $sysChange \leftarrow True$

22   **else**

23   **end**

24   **end**

25   **end**

26 **until** $(sysChange = False)$

27

28 **return** π^*

CHAPTER 5   METHODOLOGY AND DESIGN

5.1   Context and Problem

Previous works have shown that migration time can be predicted (Akoush *et al.*, 2010) and down-time optimized (Mansour *et al.*, 2016), (Li et Wu, 2016) during live migration, however, none of the available approaches have attempted to predict if a live migration would fail before migrating the VM. Moreover, to the best of our knowledge, none of the previous studies have experimented with both pre-copy, and post-copy algorithms and different types of hypervisors. The goal of this empirical study is to propose a novel approach, which uses supervised and reinforcement learning techniques that rely on the current environmental state of the system, to predict the optimal time and node, where a VM should be migrated to.

5.2   Study Definition and Design

Our empirical study aims at assessing the effectiveness of our proposed approach RISULM. We will compare the results of RISULM, with those of previous approach from the literature, i.e., MLDO and LBFT, which were proposed to predict migration time and optimize migration downtime. We will compare the performance of these approaches for both the pre-copy and post-copy algorithms, using KVM and Xen hypervisors. In the following, we introduce the specific research questions that were investigated in this study and we describe the experimental setup used to answer the research questions.

5.2.1   Research Questions

This thesis answers the following research questions:

(RQ1) Can live migration failures be accurately predicted?

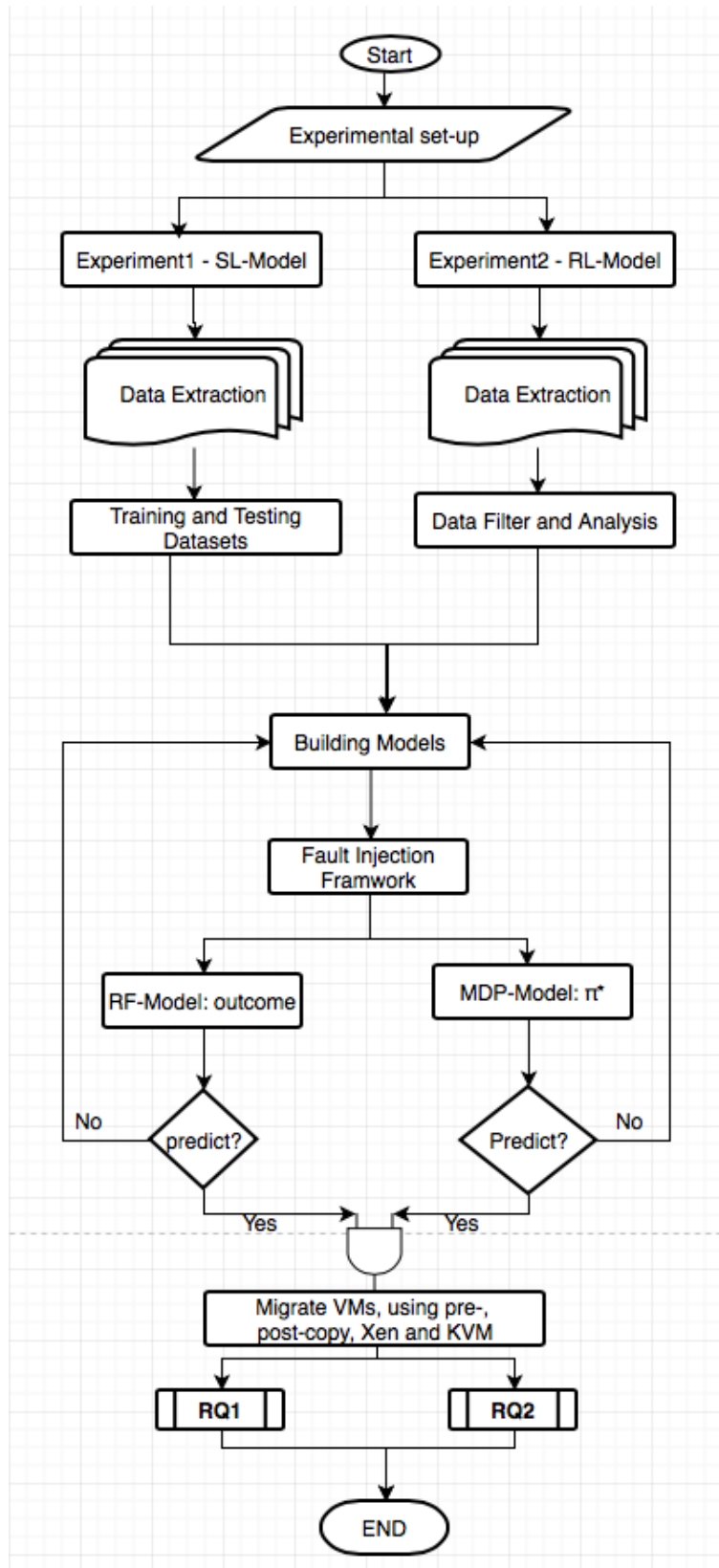**(RQ2)** Can live migration scheduling be improved using Markov Decision Processes?

Figure 5.1 Our approach, showing both models with Fault Injection Framework

5.2.2 Experimental Setup

To answer these research questions, we designed and conducted two sets of experiments. In each set of the experiment, we compared the result of our RISULM, MLDO, and LBFT against that of OpenStack using the default schedulers. We used a fault injection framework (Xiaoyong *et al.*, 2014) for OpenStack, which we customized, to randomly select one of the compute nodes and inject faults to the nodes, which will make the node to degrade up to the point of shunting down. This way we examine how our model can simulate a real world scenario, when a node becomes faulty or inconsistent in a cluster or a datacenter. In our configurations, we implemented our models in a cluster of five nodes consisting of a controller node, which is also the NFS server, and four compute nodes. We used the same topology for both experiments with 1-Controller node and 4-Compute nodes, see Figure 5.2. Our configuration allows libvirt to listen to connections on the TCP port. All our nodes have two network interface cards (NIC), one for the LAN and the other for VMs via Neutron. It is important an recommended by OpenStack to have separate network domain for VMs and nodes, when using neutron. This way, with Neutron, we can configure our Nodes on the same subnet mask, for example in private class C, and our VMs on a different network for example private class A. In our approach, we use the OpenStack (devstack) Mitaka release. We provide all the configurations and scripts used in this study, in our on-line bit bucket repository [1].

To connect all our nodes together, we used a local area network (LAN) with a high bandwidth switch (1 Gbits/s) that connects to the internet. The specifications of our system (all nodes have the same specifications) is as follows:

- CPU: Intel® Xeon® Processor E5-2690 (20M Cache, 2.90 GHz, 8.00 GTs Intel® QPI)

- RAM: 128GB

- QEMU version 2.6.0

- Libvirt v1.3.2

- Nova (Liberty) 12.0.5, with python-novaclient 2.30.2

- Hard Disk Drive (HDD) SATA : 2TB, western digital RED.

- Ubuntu 14.04

---

[1]"https://bitbucket.org/foundjem/thesis/"

Table 5.1 Flavors used in this studies.

Flavor	VCPUs	Disk(in GB)	RAM(in MB)
m1.tiny (t)	1	10	1024
m1.small (s)	1	20	2048
m1.medium (m)	2	40	4096
m1.large (l)	4	80	8192
m1.xlarge (xl)	8	160	16384
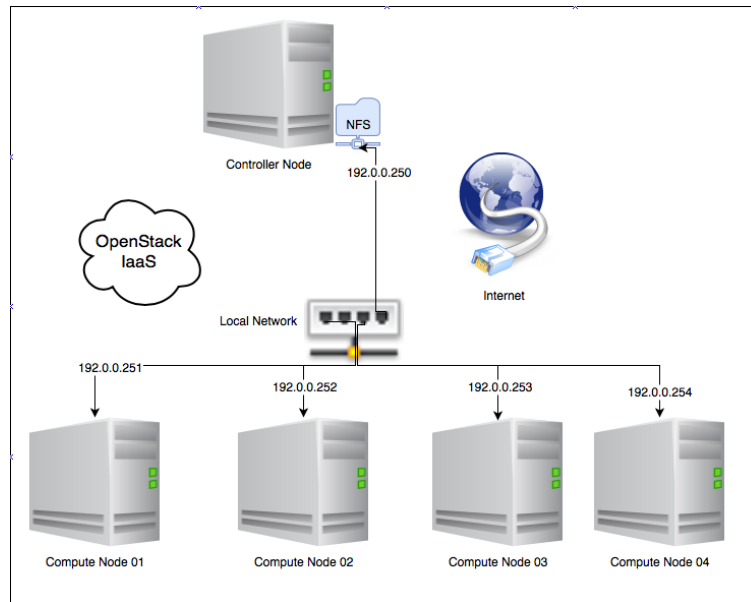
Figure 5.2 Our experimental setup and configuration

Experiment 1

In the first set of experiments which aims to investigate if we can accurately predict failures (using our random Forests model) when live migrating VMs, we first performed two sets of runs with the fault injection framework. A first run with OpenStack schedulers as shown in Figure 5.3 and the second with RISULM (implementing random forests), MLDO and LBFT models. We compare the result of these models against those of OpenStack schedulers and reports their accuracy in Figures: 6.1, 6.13, 6.14 6.15, and 6.16. For the p-values with cliff's delta effect sizes, we show our results in Tables 5.4, 5.5 and 5.6.
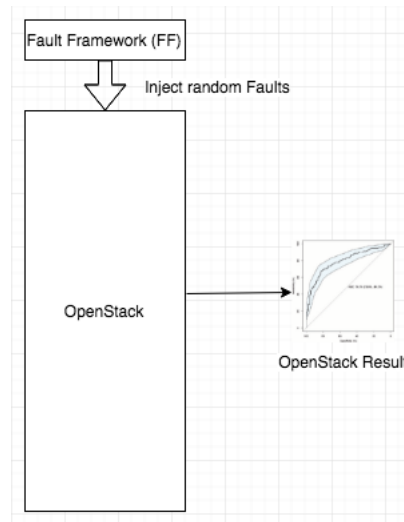
Figure 5.3 OpenStack Experimental Setup

Experiment 2

Our second set of experiments (shown in Figure 5.4), aims at investigating if an MDP can improve live migration scheduling of VMs. We implemented RISULM with and without the fault injection framework (FF). We observed the behavior of RISULM and computed its accuracy. Then, we also introduced both the MLDO and LBFT models and observed their behaviors and computed their accuracy as well. Our results are shown in Figures: 6.13, 6.14, 6.15 and 6.16.

Figure 5.4 RISULM Conceptual Architecture, showing input models and Output results

Figure 5.5 Implementation of our proposed system, Using different models and hypervisors.

5.2.3   Implementation of Models

We implemented RISULM using both the random forest and an MDP algorithm in python. RISULM is a hybrid predictive model that works as a scheduler for the live migration of VMs in OpenStack cloud. Figure 5.4 shows the architectural design of RISULM, describing how it is implemented in OpenStack cloud. We have implemented RISULM in a unified framework, co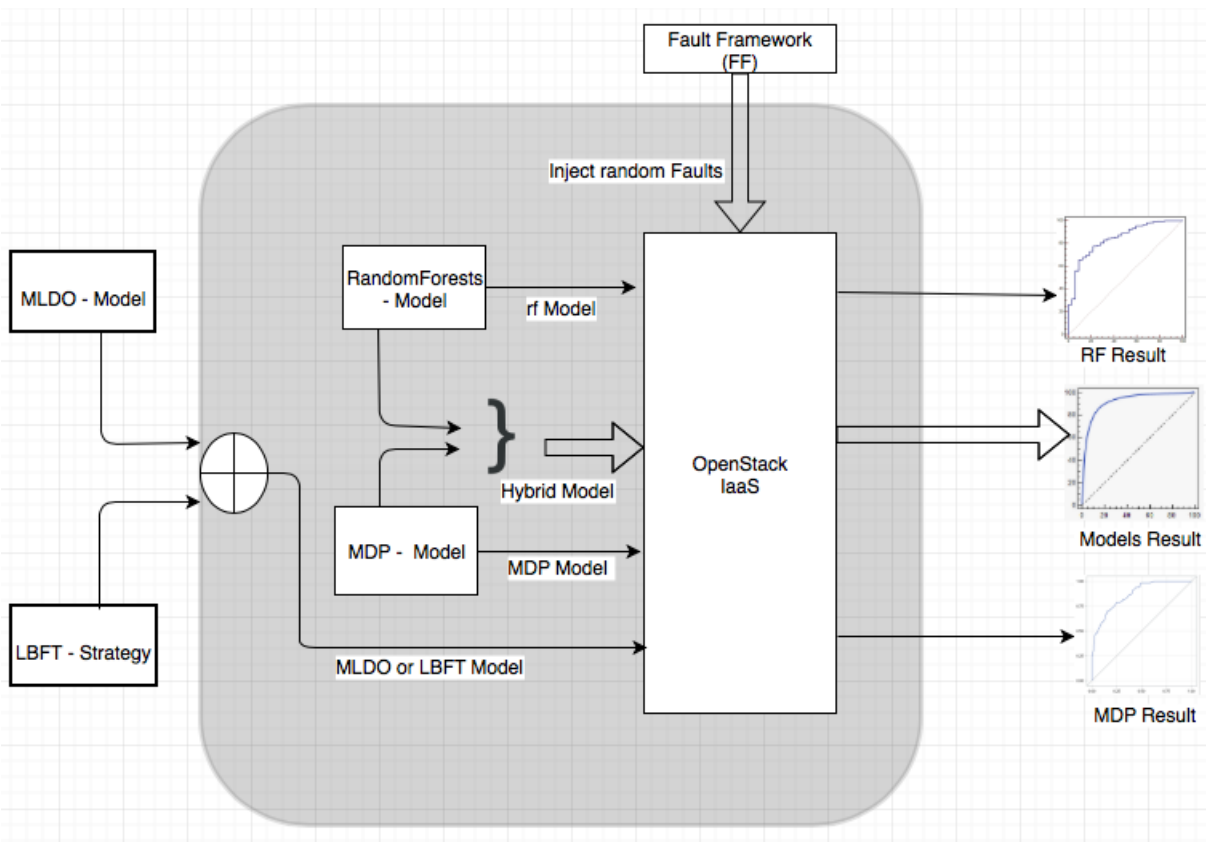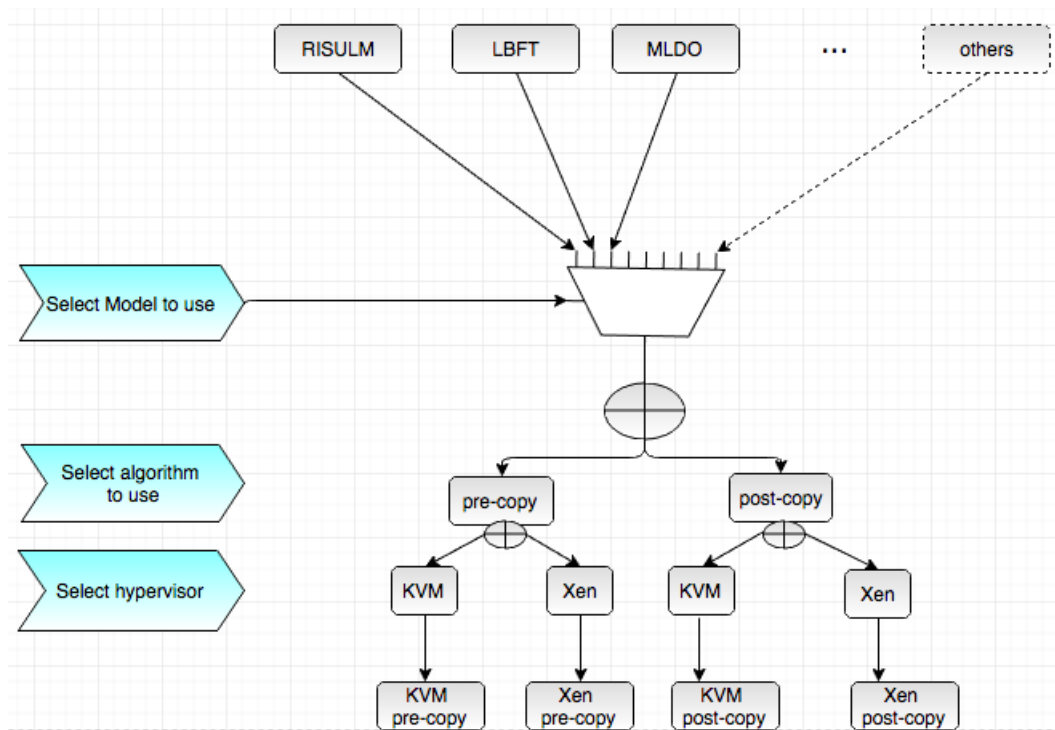nsisting of a random forest model, an MDP model, a fault framework (FF) and the native OpenStack IaaS, with possibilities for including other models, such as MLDO, LBFT, etc. This framework allows us to compare the performance of these models against the performance of the native OpenStack scheduler. Moreover, each model in our framework (such as the RF model, the MDP-model, MLDO, or LBFT) can be used separately to measure its performance against native OpenStack scheduler. Our implementation includes all four combinations of Pre-copy (KVM), Post-Copy(KVM), Pre-Copy(Xen), and Post-Copy(Xen) algorithms.

Currently, no approach exist that uses both supervised and reinforced learning model on live migration, even though machine learning have been used previously. We consider recent works in the literature that have been tested to predict live migration with high accuracy equal to or greater than 90% and can minimize migration and downtime.

After doing a survey in the literature, we chose these two models MLDO and LBFT, to compare our result with. MLDO and LBFT models happens to be the existing approaches in the literature, which have been applied on live migration studies and closely relate to our approach.

We capture the state of the machines in our cloud environment, (i.e., the VM, compute and controller nodes), by using Stress-ng (Casanovas *et al*., 2009), *Unixbench* (Mullerikkal et Sastri, 2015). We used *iperf* (Barayuga et Yu, 2015) to generate workloads on the VMs, the Controller and Compute Nodes. Then we use bash and Python scripts with regular expressions to extract data about the execution of migration operations and analyze them.

Therefore, immediately before we initiate a VM(s) for live migration, for example at any arbitrary time (t_n). The present state of the entire system, represented by these parameters in table 5.2 is known by our model, which we train dynamically. If the state of the system predicts that the live migration will not be successful, the decision to live migrate the VM will not be issued. The outcome of RISULM (using RF model) is a Boolean value (i.e., F for failure and S for success).

We launch[2] 125 VMs (that is twenty five instances for each of the five flavors: Tiny, Small, Medium, Large and xLarge) and, execute live migration with all the flavor types of VMs in our experiments.

---

[2]create VMs and keep them running on the node where they were created

In total, we execute 500^3 VMs live migration and capture the metrics values. We modified the tiny flavor of OpenStack as shown in table 5.1, by upgrading the RAM from 512 to 1GB and the Disk from 1 to 10GB. The minimum capacity that the Ubuntu 14.04 image used in this study can handle is 5G, but because of the capacity of workloads that we are running on each VM, we increase the images size of the tiny flavor from 5GB to 10GB, in this study.

Random Forests - RF model

After training and testing our models, which took about 10 min to predict live migration failures, with all the metrics values and having over 500 data points, we could now implement RISULM dynamically. That is, we captured the current state of the environment (system) and introduced it into RISULM, which predicts the outcomes dynamically as the states of the system changed over time, see Figure 4.1. Since we are observing the current state t_0 of the VMs, the controller and compute nodes to make decision if migration will be successful or not at time t_1, from a source to some destinations, where t_1 - t_0 gives us a window interval. We run a script that collects and computes the systems utilization every $2(sec)$, we chose this time window because our real-time prediction model takes approximately one sec to execute. Therefore, we decided to use a time that will not overlap with the prediction time, so we double the time to give more window for prediction. Then, we continue observing if the state of the environment (systems) if they have changed, this would enable us to take the next action.

Markov Decision Process - MDP

Using our configuration as shown in Figure 5.2, each Node represents a node where VMs could migrate from one node to another by applying an action. The state of the nodes at the present moment determines which action to take. Since we are using a shared storage location (NFS), each node is keeping track of its own environment. This way, we limit the amount of ssh client connections to each node and VMs. That is, each node periodically monitors its state and updates the current value to a data structure in the shared location, since we know the size of each node and the size of the VMs each node knows how much space is available or occupied, the CPU%, RAM%, and bandwidth currently used. The controller reads the data structure and knows which node is ready to receive VMs and the amount of VMs it can receive, depending on its available size at that moment. If more than one nodes are ready, the *comparator* decides which nodes first sent a request and respect the priority by applying an action based of first come first save policy. Our policy $\pi \leftarrow \langle VM_s, VM_d, a_1, r_1, outcome \rangle$ This policy allows us to obtain a reward r_1 when an

---

[3]125-VMs x 4-models(RISULM, MLDO,LBFT, OpenStack)

action a_1 is implemented during a live migration from a source VM_s to a destination node VM_d, and to keep track of the number of times that it would be successful. This is necessary to implement reinforcement learning.

The input to our system is the immediate environment variables such as CPU% RAM% Bandwidth etc., and we want to minimize migration time and downtime. Therefore our outputs (outcomes) are the migration time and downtime.

π represents a policy that maps a state s to an action a. Our goal using MDP is to find optimal policy that maximizes reward, as we do policy iteration.

Machine Learning based Downtime Optimization (MLDO) approach

This approach proposed by (Arif *et al.*, 2016) is similar to our random forests approach as it uses a machine learning technique to do VM live migration predictions. It implements a decision tree that decides whether or not VM should be migrated; a binary decision. However, the implementation is independent of our random forests approach. Also, MLDO uses the percentage utilization of the CPU, RAM and network (BW) to train and predict whether to live migrate VMs. That is, metrics parameters (RAM, CPU and BW) are monitored dynamically, using our setup shown in Figure 5.2, we build a scheduler using the python package "sklearn.tree.DecisionTreeClassifier" (Buitinck *et al.*, 2013). We implemented the prediction algorithm proposed by Arif et al., and computed values of the confusion metrics. We report our results in Table 6.1, which shows an accuracy of within 92%. The key idea behind MDLO is that, it uses threshold values for all the three metrics parameters with the CPU, RAM and BW arranged in the order of precedence, depending on how they affect live migration, as shown in Figure 5.6. To decide on the threshold values of our parameters, we had to run live migration several times.

Load Balance and Fault Tolerance - Strategy (LBFT)

In this implementation, we use all the five nodes in our cluster as shown in Figure 5.2, two of the nodes are loaded with CPU and memory intensive applications, we called these two nodes "hotspots" that means, some of the VMs in these nodes should be migrated to the other nodes in order to balance the work loads that the nodes are running. The main idea behind this strategy is that it uses a threshold value (on CPU and RAM) to decide on which VM should be migrated, and where. To decide on which node is less busy (i.e., CPU and RAM usage doesn't exceed the threshold value), we have implemented a shared storage (NFS) that all the nodes can access and write to.

To inject failures in the cluster, we follow the same approach as in the case of MDP. In Algorithm 3, the median values of the current resource consumption of VMs is computed regularly in, line 13,
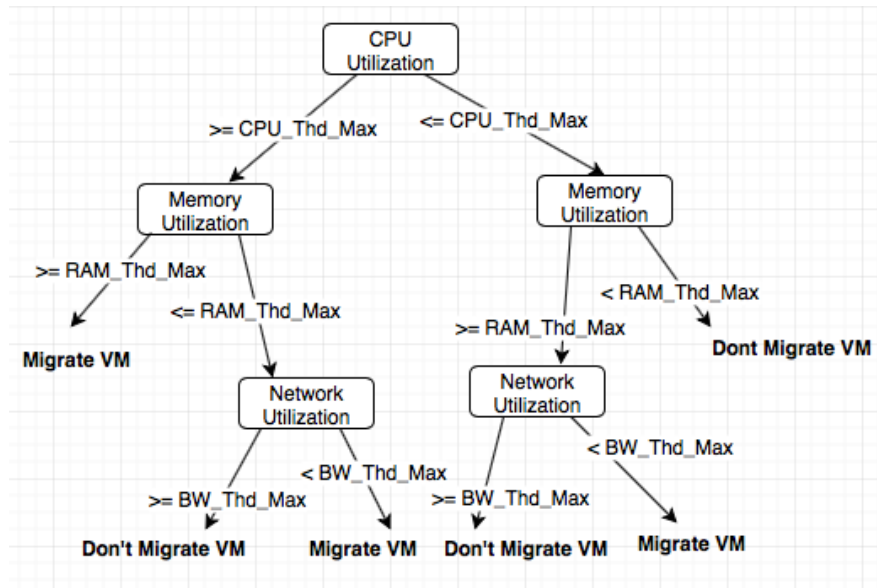
Figure 5.6 MLDO - decision tree using threshold values of metrics parameters to decide migration.

and based on that, and on the set of VMs (as stopping condition in line 14), we compute the cost function and predict the migration time. This allows us to decide on the VM that should be migrate (using the index *i* in line 18, 19 and 22).

We implemented Algorithm 3 proposed by   (Li et Wu, 2016), at the optimal time (T_{opt}), to migrate the VMs. The cost function is also computed. In line 18 of the LBFT Strategy, the predictions are computed and based on that, we computed the accuracy using the actual measured (migration) time, predicted time of each workload and each flavor (i.e., VM type) used in this study. We also computed the prediction error of the LBFT. We repeated the analysis for the two hypervisors (KVM and Xen), using both pre-copy and post-copy algorithms.

Algorithm 3: LBFT–Strategy

   Result: Migrate VMs when threshold is reached

   Input   **:**

1 Extract variables utilization that describe state of Node and running VMs

   Output   **:**

2 $VM_{migrate}$

   Local variable:

3 $VMSET \longleftarrow \{VM_1, VM_2, \ldots VM_n\}$

4 $CPU \leftarrow$ sum of *all current cpu utilization*

5 $RAM \leftarrow$ sum of *all current RAM utilization*

6 $BW \leftarrow$ sum of *all current network utilization*

7 $nodeSize \leftarrow$ updated*Nodes capacity*

8 $cost \leftarrow t_0$

9 $F \leftarrow null$

10 $index, T_a \leftarrow t_0$

11 $Thd \leftarrow minVal$

12 $outcome \leftarrow null$

13 $current\_utilization \leftarrow median(RAM, CPU, BW, VMz)$

14 **while** $VMSET \neq \emptyset \wedge current\_utilization \leq Thd$ **do**

15   **foreach** $vm \in VMSET$ **do**

16 $outcome \leftarrow model$

17   **if** $outcome$ **then**

18 $T_{opt} \leftarrow predict(migrationTimeCost)$

19 $F_{cost} \leftarrow costFunct(costPara)$

20 $VMSET \longleftarrow \emptyset$

21   **if** $F_{cost} > cost$ **then**

22 $index \leftarrow i$

23   **end**

24   **end**

25   **end**

26 $VM_{migrate} \leftarrow VMSET[i]$

27 **end**

28 **return** $VM_{migrate}$

5.2.4   Choice of Metrics

We observed from our experimentations that VMs with high CPU rate, high RAM intensity work-loads, heavy traffic on the network, high rate of dirty pages, and the size of the VM memory (VMZISE) were more likely to cause failure of VMs during live migration operation. For example, VMs with high rates of dirty pages failed to live migrate because the rate at which pages were getting dirty was higher than the copy rate. However, we also observed that when the VMs were running low RAM intensity workloads, the rate at which the pages were getting dirty was lower than the rate at which the pages were being copied, which resulted in live migration succeeding. We investigated the outcome of VMs migrations, while varying the characteristics of the VM and the workloads, and identified threshold values (expressed in term of percentage of usage of the resource that is captured by the metric) beyond which live migration always fail. We used the Spearman's rank correlation coefficient (Spearman's ρ, $-1 \leqslant \rho \leqslant 1$) to capture the correlation among the metrics, and found that only the CPU, the RAM and the Network load were not correlated. We therefore eliminated the other metrics that were strongly correlated.

Data Extraction

We mined the log files located at $/opt/stack/logs$. This location keeps an up-to date record (that is, from the first time the stack was up running till present) of all the enabled services running under OpenStack. This also includes the n-cpu.log files that keeps an up-to date record of VMs (instances) management metrics, such as the time and location of the VMs when they were created, lunched, deleted, migrated etc., all the signals captured by nova are recorded in this log file. We also have the n-net.log for networking and the like.

Using the n-cpu.log file, we were able to extract informations about the states of VMS (instances) that were created and executed on the Nodes during live migration operations (Libvirt, 2016). We used a regular expression to process the log files and *grep* specific patterns such as "PAUSED" (see Figure 5.7), since the logs are written in the order in which they occurred. We use this information to create a csv file for each instance (using *instance_id*) with their corresponding attributes such as, date in the Combined date-time in *UTC* format. We mapped the time an instance entered the stop and copy phases from the source Node A, and links this information to the same instance and the time when the VM Resumed (Life cycle Event) at the destination; Node B. Since this time is exact and it is extracted from the log files, we believe that it allows us to compute the downtime of VMs more precise, than the approach implemented by   (Salfner *et al.*, 2012), they compute the VM downtime using information about the amount of ping messages that are lost, and consider the lost pings as the downtime. However, this method turns out not to be an accurate measure for downtime

because, while using the ping command on a VM that is under live migration, from source node to destination node, there are certain intervals where the ping messages get lost whereas the VM is still on the source node and Slafner et al. did not account for these lost ping, but consider it as the total downtime of the VM under live migration. However, these lost pings are not due to the time when the VM is turn off in the source node till the time it is turn on in the destination node (downtime). Therefore, this method adds noise to the computation of downtime. In addition to computing VM downtime, we also extracted information such as data copy rate and live migration time as shown in Figure 5.7 etc. To gather information about the impact of VM workloads on the outcome of VM migrations, we executed Unixbech, sysbench and stress-ng benchmarks and collected informations on input/output write and read rate, network traffic, hard disk read/write, virtual memory rate, and faults.

To capture the impact of the network, we used a configuration that included links with two different speeds : 100Mbps and 1Gpbs. For the CPU%, RAM% and bandwidth, we used the $iftop$, top and $iperf$ tool to extract the percentage of RAM that each Node is using at any given time t_{sec}, using a regular expression. We were able to extract the total RAM%, CPU% , and bandwidth by summing up all the individual processes that were ruining at a particular time. For example, since top tools displays their values on columns (RAM, CPU, VIRT etc.) and rows (processes running) format, we extract the particular column of interest and sum all the running processes and return the sum, which we then write to the Node location of the shared memory data structure. This is updated at regular interval, and processed in real-time by RISULM.

Figure 5.7 A cross section Log file on of Node A, in VM "PAUSE" state, during live migration.

Loaded Vm in Node A, trying to migrate to Node B

memory copy rate from 100%

migration time in sec.
Memory state of Node A

Vm in Node A migrating to Node B

memory copy rate from 100%

migration time in sec.
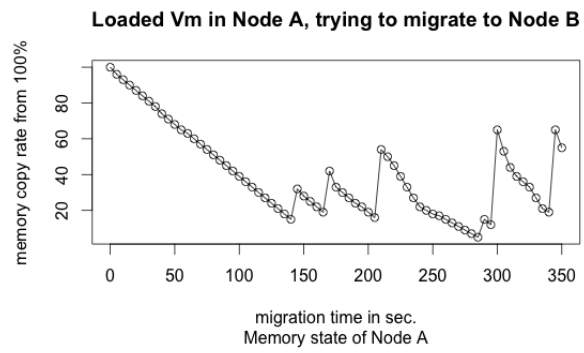Memory state of Host A

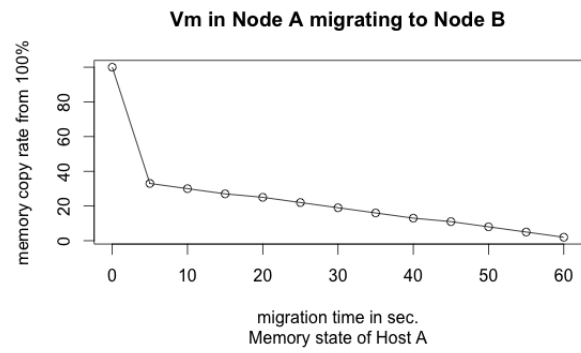Figure 5.8 Failed live-migration   Figure 5.9 Successful live-migration.

Effect of page dirtied rate on Live migration.

5.2.5   Hypotheses

To answer our research questions, we formulate the following null hypotheses:

- H_0^1 : There is no difference between the average downtime of VMs that are live migrated using the OpenStack scheduler and RISULM.

- H_0^2 : There is no difference between the average migration time of VMs when the live migration is performed using the OpenStack scheduler and RISULM.

- H_0^3 : There is no difference between the average downtime of VMs that are live migrated using the OpenStack scheduler and MLDO.

- H_0^4 : There is no difference between the average migration time of VMs when the live migration is performed using the OpenStack scheduler and MLDO.

- H_0^5 : There is no difference between the average downtime of VMs that are live migrated using the OpenStack scheduler and LBFT.

- H_0^6 : There is no difference between the average migration time of VMs when the live migration is performed using the OpenStack scheduler and LBFT.

5.2.6 Analysis Method

We use the Mann-Whitney U test to test our null Hypotheses H_0^x, for $x \in \{1, 2, \ldots 6\}$. The Mann-Whitney U test is a non-parametric statistical test used for assessing whether two independent distributions have equally large values. Non-parametric statistical methods make no assumptions about the distributions of the assessed variables. We set α to 0.05. Moreover, we consider a Mann-Whitney test result to be statistically significant if and only if the *p*-value is below α. Additionally, we compute the effect-size of the difference using Cliff's δ (cohen 1998) (Macbeth *et al.*, 2011), which is also a non-parametric statistic test, effect size measure, which measures how often values in one distribution are larger than values in another distribution. Cliff's δ or d values lies between [-1,1] and is considered small for $0.148 \leq d < 0.33$, medium for $0.33 \leq d < 0.474$, and large for $d \geq 0.474$. Our results are reported in tables 5.4, 5.5, 5.4, and 5.6. We report only significant *p*-values for our studies.

Furthermore, we used the Spearman's rank correlation coefficient to capture the correlation among our different variables. This a non-parametric measure of rank correlation that measures statistical dependence between the ranking of two variables. It assesses how well the relationship between two variables can be described using a monotonic function. Mukaka et al. explain how to interpret the values of ρ (Mukaka, 2012).

Table 5.2 Metrics used in our study to predict VMs live migration and the rationales.

Metrics of Our Study			
Metrics	Source Node	Destination Node	Virtual Machine
RAM%	**H**ost running memory intensive applications are likely to fail during live migration operations in OpenStack.	**T**he memory state of the destination machine can determine if the VM will success or not and also, how long it will take to complete migration.	**VMSIZE**, running memory intensive application have high changes of failure, that is directly proportional to the size of the VM.
CPU%	**I**ntensive workload may have scheduling problem and delay live migration.	**I**ntensive workload on the destination, will prolong migration time.	**vCPU**, intensive applications running on a live migrated VM may disrupt it state [4]
Network%	**B**andwidth available during live migration might affects downtime and migration time.	**B**andwidth congestion of destination node might affects live migration time and downtime.	**B**andwidth available for VM, might affects the applications running on it ans might also affects downtime and migration time.
Page Dirtying Rate %	**H**igher rate have high chances of failure, during iterative pre-copy phase.[5]	**H**owever, this might increase migration time as more pages get dirtied in Node A.	**P**ages copied until the rate of copying becomes more than rate of dirtying. This might affects stopping condition and fails migration.

Table 5.3 Metrics used to determine live migration of VMs outcome and description.

Metrics	Description
Migration Time	Total time T_M(sec) taken to live migrate a VM from a Node A to a Node B. This time starts from the initialization of the VM live migration process to the moment when the VM starts running on the Node B.
Downtime	During the Stop and copy phases as explained in both the pre-, and post-copy algorithms, refers to the time when the VM is suspended in Node A and resumes on Node B. For optimal operation we need to keep this time to its minimal.

Table 5.4 *p*-value and cliff's-δ showing results of significant p-values for **Migration time**, **RISULM** against OpenStack Scheduler with different hypervisors and algorithms.

Flavors	Pre-Copy KVM		Post-Copy KVM		Pre-Copy Xen		Post-Copy Xen	
	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ
Tiny	$2.101x10^{-06}$	large	$1.250x10^{-08}$	large	$1.140x10^{-05}$	large	$1.240x10^{-06}$	large
Small	$2.123x10^{-06}$	large	$1.206x10^{-08}$	large	$1.310x10^{-05}$	large	$1.480x10^{-06}$	large
Medium	$3.148x10^{-06}$	large	$1.185x10^{-08}$	large	$1.515x10^{-05}$	large	$1.619x10^{-06}$	large
Large	$1.501x10^{-06}$	large	$1.421x10^{-09}$	large	$1.701x10^{-05}$	large	$1.872x10^{-07}$	large
xLarge	$1.731x10^{-06}$	large	$1.578x10^{-09}$	large	$1.813x10^{-05}$	large	$1.990x10^{-07}$	large

Table 5.5 *p*-value and cliff's-δ showing results of significant p-values for **Migration time**, **MLDO** against OpenStack Scheduler with different hypervisors and algorithms.

Flavors	Pre-Copy KVM		Post-Copy KVM		Pre-Copy Xen		Post-Copy Xen	
	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ
Tiny	$2.250x10^{-05}$	large	$2.101x10^{-07}$	large	$4.740x10^{-05}$	large	$2.211x10^{-06}$	large
Small	$2.320x10^{-05}$	large	$2.232x10^{-07}$	large	$4.940x10^{-05}$	large	$2.275x10^{-06}$	large
Medium	$1.115x10^{-05}$	large	$2.341x10^{-07}$	large	$4.995x10^{-05}$	large	$2.104x10^{-06}$	large
Large	$1.553x10^{-06}$	large	$2.481x10^{-08}$	large	$5.142x10^{-05}$	large	$1.117x10^{-06}$	large
xLarge	$1.866x10^{-06}$	large	$2.663x10^{-08}$	large	$5.013x10^{-05}$	large	$1.654x10^{-06}$	large

Table 5.6 *p*-value and cliff's-δ showing results of significant p-values for **Migration time**, **LBFT** against OpenStack Scheduler with different hypervisors and algorithms.

Flavors	Pre-Copy KVM		Post-Copy KVM		Pre-Copy Xen		Post-Copy Xen	
	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ
Tiny	$4.343x10^{-04}$	Medium	$4.376x10^{-06}$	large	$4.401x10^{-04}$	Medium	$4.248x10^{-05}$	large
Small	$4.305x10^{-04}$	Medium	$4.310x10^{-06}$	large	$4.521x10^{-04}$	Medium	$4.534x10^{-05}$	large
Medium	$2.535x10^{-04}$	Medium	$2.485x10^{-06}$	large	$2.745x10^{-04}$	Medium	$4.618x10^{-05}$	large
Large	$2.901x10^{-05}$	large	$2.898x10^{-07}$	large	$2.894x10^{-05}$	large	$4.957x10^{-06}$	large
xLarge	$2.881x10^{-05}$	large	$2.878x10^{-07}$	large	$2.713x10^{-05}$	large	$4.597x10^{-06}$	large

Table 5.7 *p*-value and cliff's-δ showing **Downtime** results of significant p-values for **RISULM** against OpenStack with KVM, Xen, using pre-, and post-copy algorithm.

Flavors	Pre-Copy KVM		Post-Copy KVM		Pre-Copy Xen		Post-Copy Xen	
	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ
Tiny	$8.341x10^{-06}$	large	$8.014x10^{-06}$	large	$8.625x10^{-06}$	large	$8.521x10^{-06}$	large
Small	$6.224x10^{-06}$	large	$6.545x10^{-06}$	large	$7.332x10^{-06}$	large	$2.320x10^{-06}$	large
Medium	$6.512x10^{-06}$	large	$2.445x10^{-06}$	large	$4.515x10^{-06}$	large	$3.237x10^{-06}$	large
Large	$5.409x10^{-06}$	large	$2.011x10^{-06}$	large	$4.611x10^{-06}$	large	$3.012x10^{-06}$	large
xLarge	$5.029x10^{-06}$	large	$2.058x10^{-06}$	large	$6.310x10^{-06}$	large	$2.072x10^{-06}$	large

Table 5.8 *p*-value and cliff's-δ showing **Downtime** results of significant p-values for **MLDO** against OpenStack with KVM, Xen, using pre-, and post-copy algorithm.

Flavors	Pre-Copy KVM		Post-Copy KVM		Pre-Copy Xen		Post-Copy Xen	
	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ
Tiny	$3.352x10^{-05}$	large	$2.108x10^{-07}$	large	$3.541x10^{-05}$	large	$3.521x10^{-06}$	large
Small	$2.551x10^{-05}$	large	$1.987x10^{-07}$	large	$2.019x10^{-05}$	large	$1.782x10^{-06}$	large
Medium	$2.971x10^{-05}$	large	$3.490x10^{-07}$	large	$2.210x10^{-05}$	large	$3.341x10^{-06}$	large
Large	$3.051x10^{-06}$	large	$3.018x10^{-08}$	large	$4.701x10^{-05}$	large	$3.902x10^{-06}$	large
xLarge	$2.131x10^{-06}$	large	$2.921x10^{-08}$	large	$3.333x10^{-05}$	large	$2.990x10^{-06}$	large

Table 5.9 *p*-value and cliff's-δ showing **Downtime** results of significant p-values for **LBFT** against OpenStack with KVM, Xen, using pre-, and post-copy algorithm.

Flavors	Pre-Copy KVM		Post-Copy KVM		Pre-Copy Xen		Post-Copy Xen	
	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ	p-value	Cliff's δ
Tiny	$1.481x10^{-04}$	Medium	$1.233x10^{-04}$	Medium	$1.602x10^{-04}$	Medium	$1.098x10^{-04}$	Medium
Small	$1.322x10^{-04}$	Medium	$1.012x10^{-04}$	Medium	$2.319x10^{-04}$	Medium	$1.663x10^{-04}$	Medium
Medium	$1.215x10^{-04}$	Medium	$1.549x10^{-04}$	Medium	$3.022x10^{-04}$	Medium	$1.308x10^{-04}$	Medium
Large	$2.901x10^{-04}$	Medium	$1.424x10^{-04}$	Medium	$2.894x10^{-04}$	Medium	$2.115x10^{-04}$	Medium
xLarge	$2.881x10^{-04}$	Medium	$2.318x10^{-04}$	Medium	$2.713x10^{-04}$	Medium	$2.097x10^{-04}$	Medium

To draw and compute the area under the ROC curve, we use information from Table 5.10, where TP, FP, TN, and FN are true positives, false positives, true negatives, and false negatives, respectively. In tables 5.11, 5.12, 5.13, and 5.14, we show the respective confusing metrics for all combinations of Algorithms and hypervisors.

Table 5.10 Evaluation indicators of our Studied Parameters.

Indicator	Expression	Description
True Negative Rate (TNR)	$\dfrac{TN}{(FP+TN)}$	the percentage of correctly predicted migration as fail
Negative Predictive Value (NPV)	$\dfrac{TN}{(FP+TN)}$	when migration is predicted fail, what proportion are wrong?
Positive Predictive Value (PPV)	$\dfrac{TP}{(TP+FP)}$	When migration is predicted successful, what proportion is correct?
True Positive Rate (TPR)	$\dfrac{TP}{(TP+FN)}$	the percentage of correctly predicted migration as successful
False Positive Rate (FPR)	$\dfrac{FP}{(FP+TN)}$	migration predicted unsuccessful, how often is migration unsuccessfully?
Accuracy (ACC)	$\dfrac{(TN+TP)}{(TP+FP+TN+FN)}$	Overall, how often does our model classify correctly?

Table 5.11 The confusion metrics of RISULM Using KVM pre-copy.

	Actual migration		
Accuracy: 0.95	Positive (P)	Negative (N)	
Positive (P)	TP = (67)	FP ($\alpha = 2$)	↔ PPV (0.97)
Negative (N)	FN ($\beta = 3$)	TN (28)	↔ NPV (0.90)

Predicted migration

TPR (0.96) TNR (0.93)

Table 5.12 The confusion metrics of RISULM Using KVM post-copy.

	Actual migration		
Accuracy: 0.95	Positive (P)	Negative (N)	
Positive (P)	TP = (68)	FP ($\alpha = 4$)	↔ PPV (0.94)
Negative (N)	FN ($\beta = 1$)	TN (27)	↔ NPV (0.96)

Predicted migration

TPR (0.99) TNR (0.87)

Table 5.13 The confusion metrics of RISULM Using Xen pre-copy.

	Actual migration		
Accuracy: 0.95	Positive (P)	Negative (N)	
Positive (P)	TP = (66)	FP ($\alpha = 2$)	↔ PPV (0.97)
Negative (N)	FN ($\beta = 3$)	TN (29)	↔ NPV (0.91)

Predicted migration

TPR (0.96) TNR (0.94)

Table 5.14 The confusion metrics of RISULM Using Xen post-copy.

	Actual migration		
Accuracy: 0.95	Positive (P)	Negative (N)	
Positive (P)	TP = (67)	FP ($\alpha = 4$)	↔ PPV (0.94)
Negative (N)	FN ($\beta = 1$)	TN (28)	↔ NPV (0.97)

Predicted migration

TPR (0.99) TNR (0.88)

5.2.7 Independent Variables

The parameters in Table 5.2, i.e., the CPU, RAM, Network utilization and the dirty page rate in Table 5.3 of the Nodes and VMs are the Independent variables of this study. The rationale behind choosing these metrics is that, they form the major factors that affect the performance of VMs under live migration, which determines if the VMs would migrate successfully or not, and also the amount of time it will take to migrate instances of VMs across the cloud.

5.2.8 Dependent Variables

The dependent variables of this study are shown in Tables 6.1, and 5.3. They measures the accuracy of our predictions of VMs live migration failures, the total migration time and the downtime of VMs during live migration. These metrics allow us to assess the effectiveness of our proposed approach, since we aim at reducing the total time it takes to migrate VMS across the cloud and to keep the downtime as smaller as possible.

CHAPTER 6 EVALUATION OF OUR PREPOSED APPROACH RISULM

To assess the effectiveness of our proposed approach (RISULM), we compare its performance against the OpenStack native scheduler, MLDO, and LBFT.

6.1 Results of Experiment 1

In this section, we report the result obtained for RQ1. These results suggest that, RF model can predict failures of VM (i.e., if a VM live migration will be successful or not) with an accuracy of 95%. Our MDP, which we use in RQ2 give an accuracy of 95%.

Furthermore, we observed that for the models with higher accuracy, the total migration time and downtime were also lower than those with lover accuracy. Therefore, based on the obtained *p*-values and large effect sizes, we can therefore reject the null hypotheses that there is no difference between migration time and downtime using OpenStack schedulers and RISULM (sl-model). Moreover, the accuracy of our prediction model (sl-model) is higher than the accuracy of Open-Stack native scheduler, as well as those of MLDO and LBFT.

Summary RQ1: In this section, we have shown that we can accurately predict live migration failures using a Random Forest model. Moreover, we were able to predict live migration failure with an accuracy of 95%, which intrinsically reduces both the total migration time and downtime, as shown in Table 6.1.

6.2 Results of Experiment 2

In this section, we report the result obtained for RQ2. Without injecting the FF, the MDP had an accuracy of 94% see figure 6.14, and with the FF injected, we observe 95% accuracy see figure 6.15. Moreover, we also observe p-values that are statistically significant with large effect size as shown in tables 5.4, 5.5, 5.6, 5.7, 5.8, and 5.9.
Therefore, we rejected the null hypotheses (H_0^1, ... H_0^6.) that there are no significant difference on average migration time and downtime using RISULM, MLDO, LBFT against OpenStack schedulers on live migration. As we were able to observe a statistically significant differences on average migration time and downtime for RISULM (H_0^1, H_0^2), MLDO (H_0^3, H_0^4) and LBFT (H_0^5, H_0^6) We also observe that, as the accuracy of MDP goes up from 94% to 95%, the accuracy of RISULM also goes up from 95% to 95.1%, whereas the random forests model (rf-model) did not notice any change, since the RISULM uses both rf-model and MDP models, we concluded that the MDP

model is responsible for this overall increase in accuracy, because we used the same treatment (injecting the FF) to the system.

Summary RQ2: Our findings shows that, using MDP, we can accurately improve where our VMs will be live migrated within an accuracy of 95.1%, we use table 5.10 to compute and show the accuracy of our study in table 6.1. We also observed that, the MDP model gives best accuracy when the system state changes more at node (datacenter) level rather than within the individual VMs.

6.2.1 Evaluation of RISULM

In this section, we discuss how our proposed model (RISULM) was evaluated. (i) We compare the performance of RISULM (especially the accuracy on predicting failures of live VMs during migration, the total migration time, and the downtime) against OpenStack native scheduler. (ii) We also use existing approaches from the literature (MLDO and LBFT) and compare their performance against OpenStack scheduler, (iii) then we analyze the performance of RISULM, MLDO, and LBFT against OpenStack with different hypervisors using both the pre-copy, and post-copy algorithms.

First and foremost, RISULM uses both SL and RL to make decision, firstly, we run an experiment with SL (rf-model) which uses random Forests. We did this to understand how accurate our model can predict failures of live VMs before we migrate them. Secondly, we run an experiment with the MDP implementation, which enable us to make timely decisions on where and when to migrate live VMs. Thirdly, we combined both random Forests model (RF - model) and the MDP - Model, which forms a hybrid model (RISULM), to understand if the performance of the MDP could improve the performance of RISULM.

The output of both models (rf-model and MDP) are Boolean ("YES", "NO"), thereafter, we combined both outputs using a conservative method to build RISULM. Conservative in the sense that, we want to maximize the possibility of success (YES, YES) before we migrate live VMs, which is why we use a logical AND gate that consumes both outputs of the rf-model and the MDP to output either a "YES" or a "NO". Keep in mind that since there are two possible inputs (N –number of inputs) into the AND gate, there ought to be $2^N - 4$ possible conditions into the AND gate, which then outputs either a YES or a NO. This way, we are sure that both models (rf-model and the MDP) should agree with a YES each before migrating live VMs.

In our experiments, we use the stress-ng tool (Casanovas *et al.*, 2009). With these tools, we set various categories of workload on the VMs and Nodes before migrating them. Base on our ob-

servations during the initial phase of our experiment, while trying all possible cases of failures and successes of live migrating VMs. We did several combinations of workloads and found that, we could categorize the VMs workload such as idle, 25%, 50%, 75%, and 100% load conditions. However, we observe that, when the VMs are loaded above 75% workload they always fail live migration, whereas below 25% of workload they always success live migration under stable network conditions.

Therefore, we consider 75% as full loaded condition, 50% as average loaded conditions and 25% as unloaded condition (idle) of the VM size (VMSIZE). In our experimental setup, all the nodes are connected through a 1Gbps link (switch), to implement the models in this study; OpenStack scheduler, MLDO, LBFT, and RISULM, see Figures 5.2 and 5.5; using KVM and Xen hypervisors with both post-copy and pre-copy algorithms. Values for migration time, downtime and data transfer rates were also recorded on all our experimental setup and as shown in Figure 6.5, 6.6, 6.7 , and 6.7; for migration time, Figure 6.9, 6.10, 6.11, and 6.12; for downtime, and Figure 6.1, 6.2, 6.3, and 6.4; for data transfer rates. We also tested our models on the effects that latency and link speed has on both the downtime and migration time on VMs during live migration see Figures: 6.25, 6.26, 6.27, and 6.28; for migration time and 6.29, 6.30, 6.31, and 6.32; for downtime. On link speed,Figures: 6.17, 6.18, 6.19, and 6.20; for migration time and Figures: 6.21, 6.22, 6.23, and 6.24; for downtime.

(i) - **RISULM against OpenStack:** We did live migration on all our 125 VM instances using OpenStack scheduler with the FF injected see Figure 5.3, we computed the performance of the studied metrics, we computed an accuracy (ACC) of 83% and record our findings in Table 6.1. The rationale for doing this is to know how OpenStack native schedulers perform on live migration, in terms of migration time and downtime. Once these values are known, we can now build our own scheduler aiming to improve the observed results.

Similarly, did live migration on all our 125 VM instances, using our proposed approach –RISULM as explained above; implementing the Algorithms 1 and 2. Even though we observe tremendous improvements with our proposed model; RISULM, we computed the performance of the studied metrics, we computed an accuracy (ACC) of 95% see Figure 6.16. We record our findings in Table 6.1, we did not just conclude at this point on the performance of RISULM, however, we needed more evidences based on known approaches that have good performances on live migration, which is why we went one step forward to implement MLDO and LBFT.

(ii) - **MLDO against OpenStack:** We ran MLDO against OpenStack native scheduler, using 125 VM instances. We maintained the same experimental conditions and noticed that MLDO performs better that OpenStack native scheduler but it is less performant than RISULM, we record our find-

ings in Table 6.1.

(iii) - **LBFT against OpenStack:** Last, we ran LBFT strategy against OpenStack native scheduler, we observe that it can achieve an accuracy of 90%, see Table 6.1. We computed the metrics values for this study as in the other models. However, LBFT performs less than MLDO and RISULM but better than OpenStack scheduler.

Both LBFT and MLDO were originally designed for different uses cases on live migration, and both perform better than OpenStack as expected, and since all the models used in this study were subjected to the same treatments, we could now conclude that, RISULM outperforms prior models that aim at optimizing migration time and down time on VMs during live migration, in a wider range of use cases spanning those of MLDO and LBFT. Also, our predicting model has the best prediction so far in the literature 95% accuracy. Moreover, our approach uses different types of hypervisors and both the pre-copy and post-copy algorithms. This is a novel approach that suggests the categories of applications or workload to use during live migration and on which specific hypervisors (KVM, Xen, etc.,) and algorithm (pre-copy or post-copy) should be chosen.

6.2.2 Scalability

We executed our proposed model (RISULM) against OpenStack native scheduler, MLDO, and LBFT on 3 to 15 nodes (3 and 15 nodes, which were chosen arbitrary to test for scalability.) on WAN links using AWS EC2[1] extra large instances. We set up 30, 60 and 150 VMs. We aim at finding how scalable RISULM could be; that is, when Nodes are added to or reduced from the pool of available Nodes. RISULM turns out to be scalable, which can handle a variable number of Nodes and VMs in a datacenter. In all runs, our results were consistent to that of our experimental set-up reported in the results section.

6.3 Discussion of our Result

For both the downtime and migration time computed, we observe a statistically significant difference using RISULM, MLDO, and LBFT. Moreover, we found a decrease of migration time and downtime with RISULM, MLDO, and LBFT against the OpenStack scheduler. Also, the performance of these models vary depending on the choice of the hypervisor used and the algorithm, this is true for all types of flavors and workloads respectively. On average, the KVM had higher

---

[1] $http://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_{Get}Started.html$

Table 6.1 Contingency table showing results of the studied Models.

Models	*Metrics*	Type of hypervisor and Algorithm			
		Pre-copy(KVM)	Post-copy(KVM)	Pre-copy(Xen)	Post-copy(Xen)
RISULM	ACC	0.95	0.95	0.95	0.95
	TPR	0.96	0.99	0.96	0.99
	TNR	0.93	0.87	0.94	0.88
	PPV	0.97	0.94	0.97	0.94
	NPV	0.90	0.96	0.91	0.97
MLDO	ACC	0.92	0.85	0.92	0.85
	TPR	0.92	0.91	0.93	0.95
	TNR	0.93	0.74	0.90	0.68
	PPV	0.97	0.87	0.96	0.83
	NPV	0.82	0.81	0.84	0.89
LBFT	ACC	0.90	0.88	0.90	0.88
	TPR	0.94	0.97	0.96	0.97
	TNR	0.81	0.71	0.78	0.71
	PPV	0.92	0.86	0.90	0.86
	NPV	0.86	0.93	0.89	0.93
OpenStack	ACC	0.83	0.76	0.81	0.74
	TPR	0.85	0.88	0.84	0.84
	TNR	0.77	0.54	0.74	0.56
	PPV	0.91	0.78	0.90	0.77
	NPV	0.65	0.70	0.63	0.67
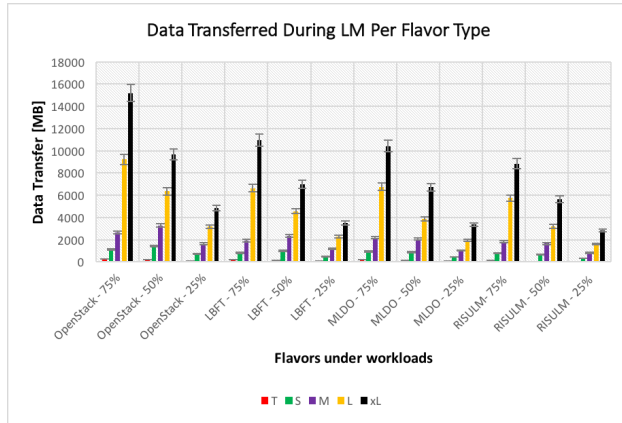
Figure 6.1 Pre-copy with KVM
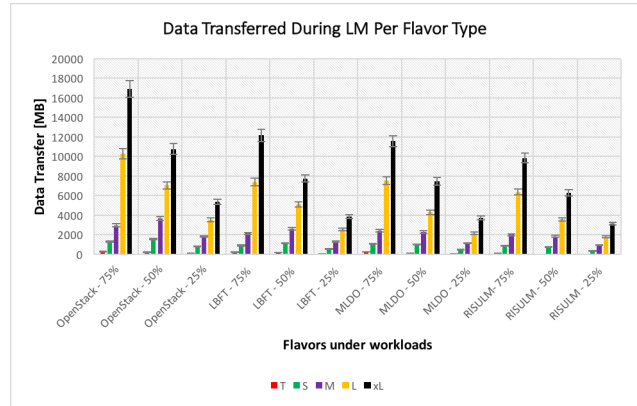
Figure 6.3 Pre-Copy with Xen.
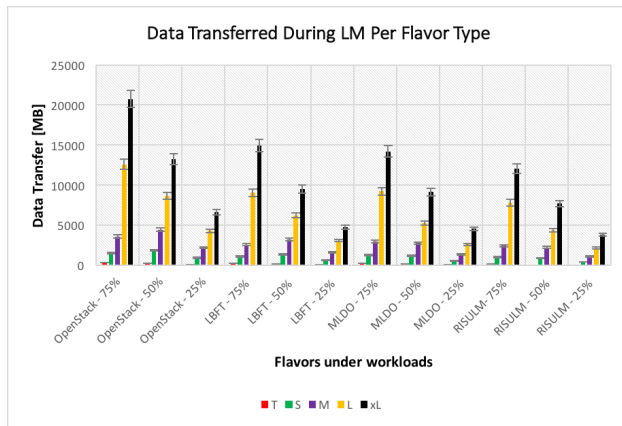
Figure 6.2 Post-copy with KVM
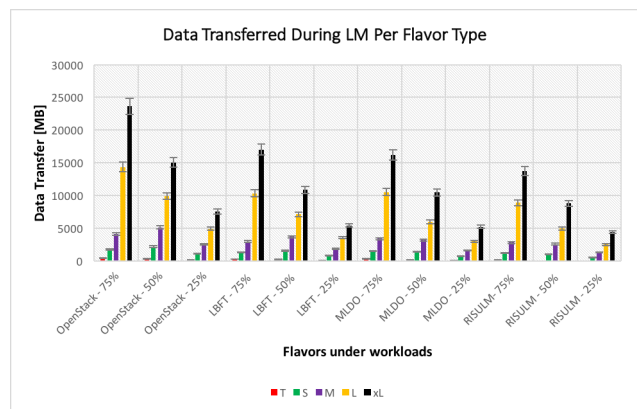
Figure 6.4 Post-Copy with Xen.

Data Transfer Rate of Models using Pre-, Post-copy Algorithm in KVM and Xen
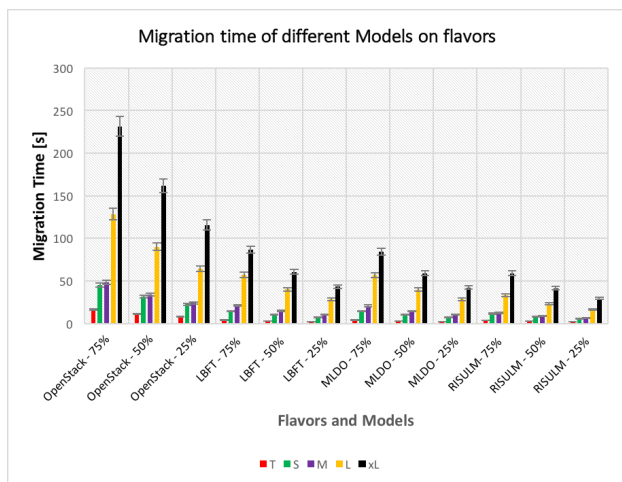
Figure 6.5 Post-copy with KVM
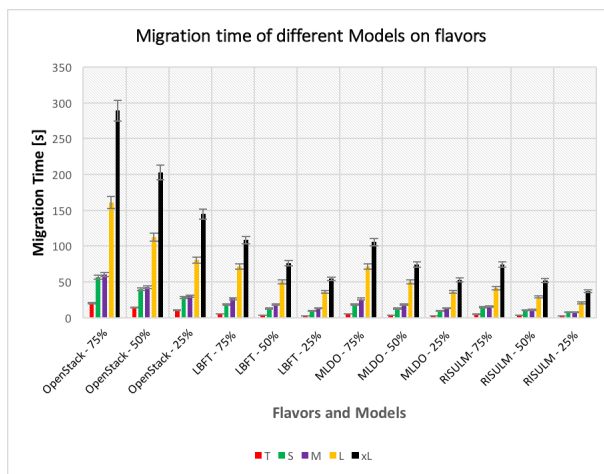
Figure 6.7 Pre-Copy with KVM.
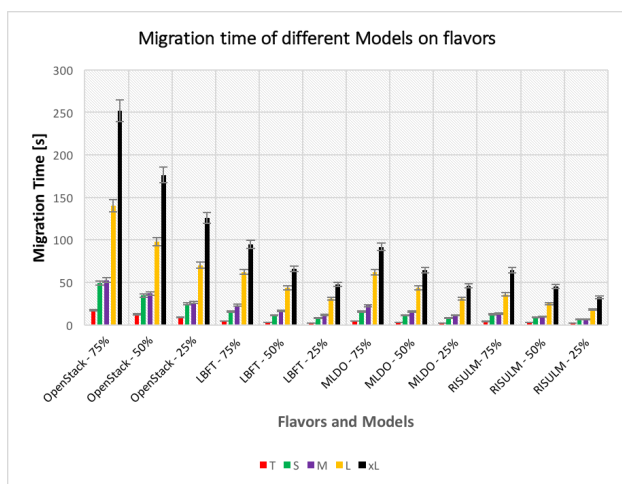
Figure 6.6 Post-copy with Xen
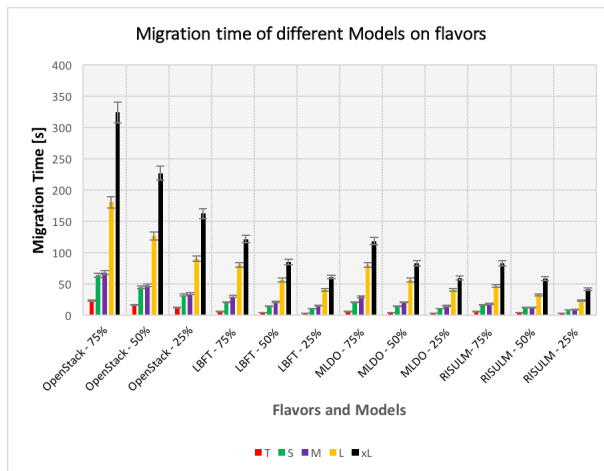
Figure 6.8 Pre-Copy with Xen.

Migration Time of VMs, different Flavors and Models using Pre-, Post-copy Algorithms
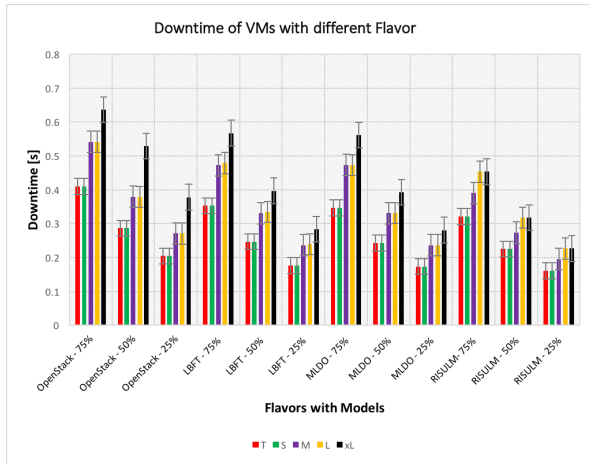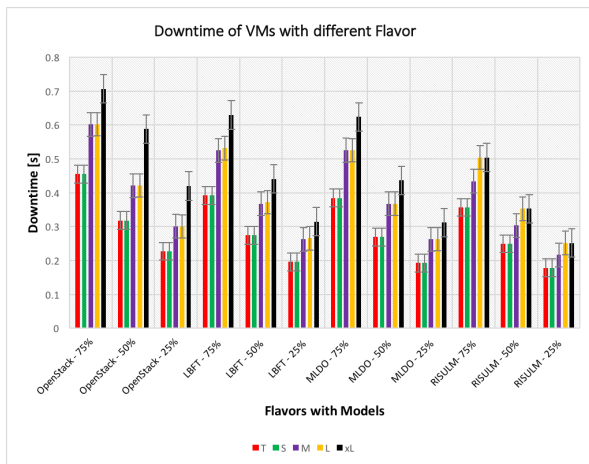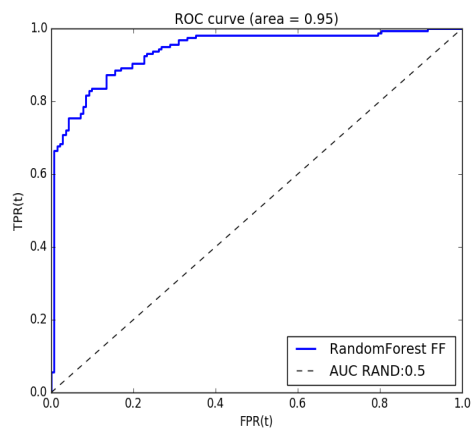
Figure 6.9 Post-copy with KVM
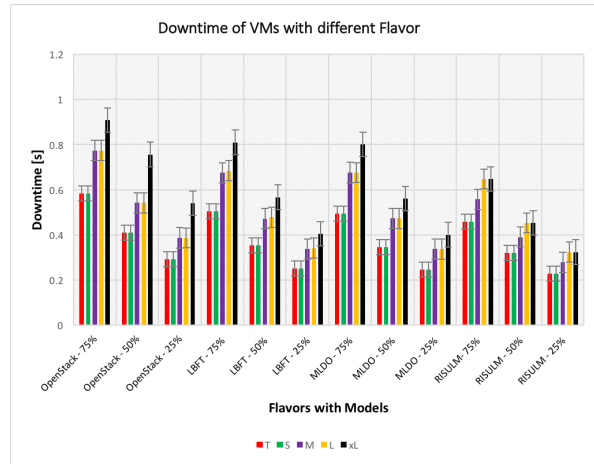
Figure 6.11 Pre-Copy with KVM.

Figure 6.10 Post-copy with Xen
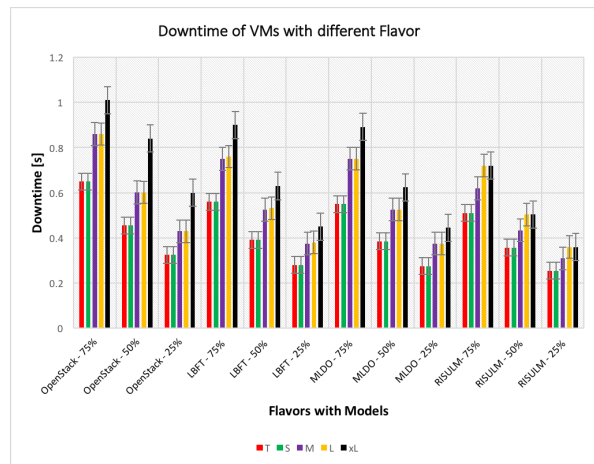
Figure 6.12 Pre-Copy with Xen.

Downtime of VMs, different Flavors and Models using Pre-, Post-copy Algorithms

Figure 6.13 Accuracy of rf-model before and after FF.
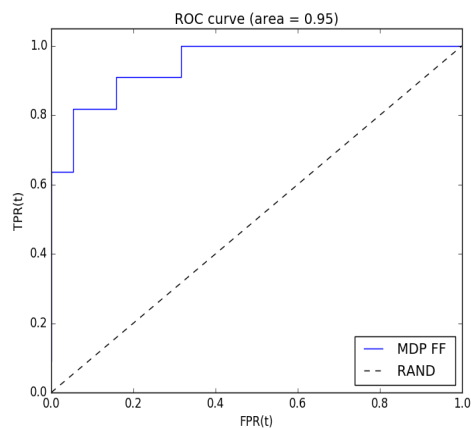
Figure 6.15 Accuracy of MDP after.
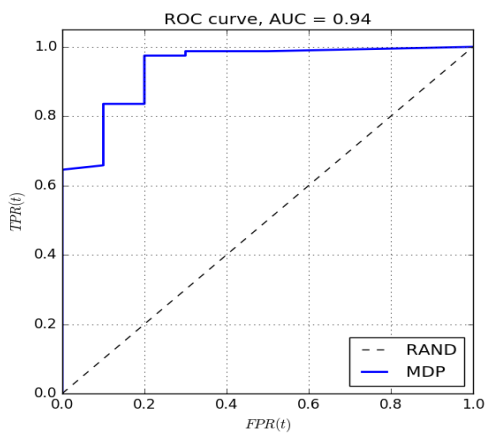
Figure 6.14 Accuracy of MDP before
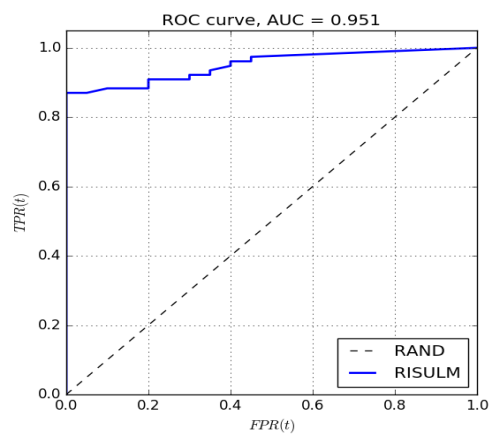
Figure 6.16 RISULM Optimal Result
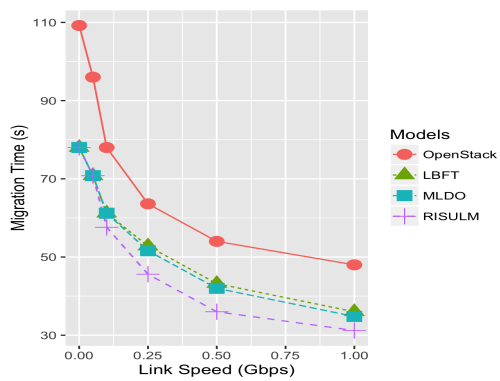
Results showing RISULM improvement by MDP
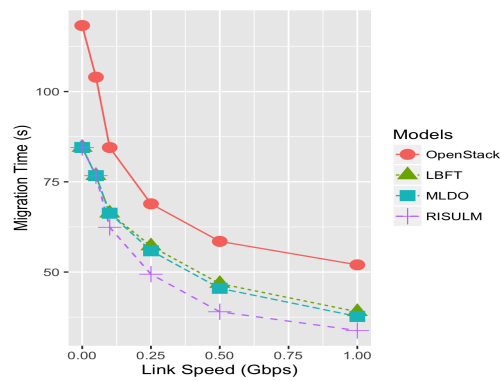
Figure 6.17 Pre-Copy KVM Models.
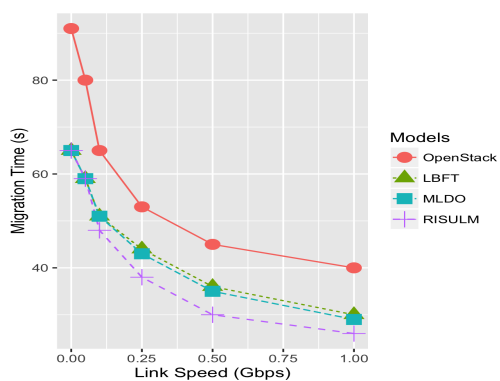

Figure 6.19 Pre-Copy Xen Models.

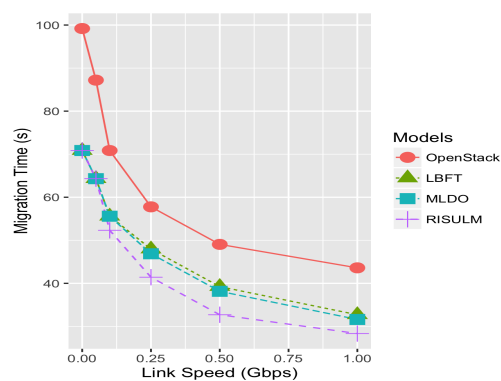Figure 6.18 Post-Copy KVM Models.

Figure 6.20 Post-Copy Xen Models.

Effect of Link speed on Migration time

Figure 6.21 Pre-Copy KVM Models.

Figure 6.23 Pre-Copy Xen Models.

Figure 6.22 Post-Copy KVM Models.

Figure 6.24 Post-Copy Xen Models.

Effect of Link speed on Downtime.
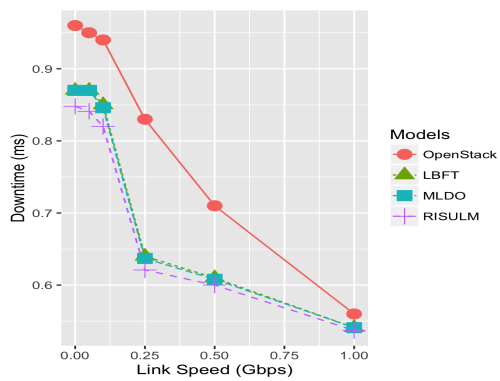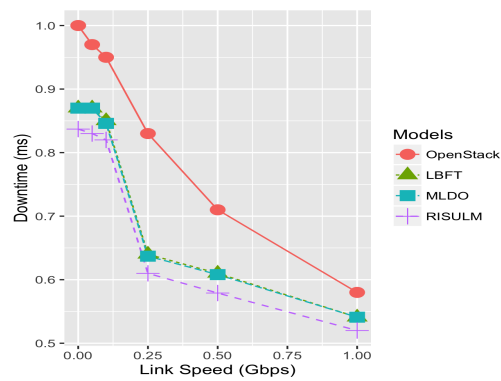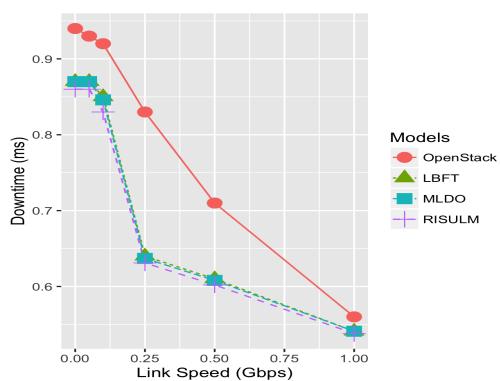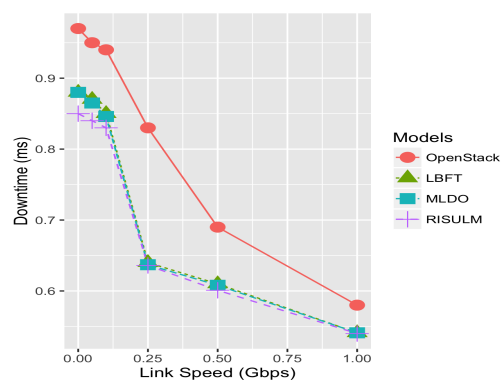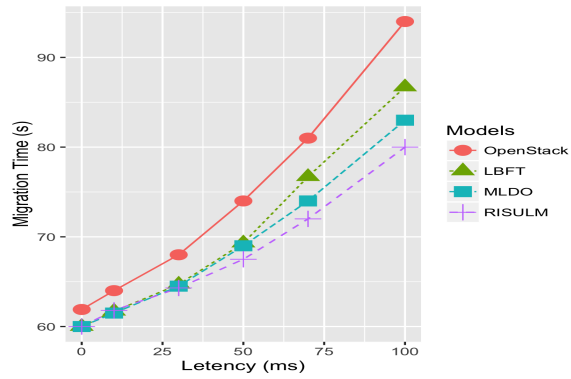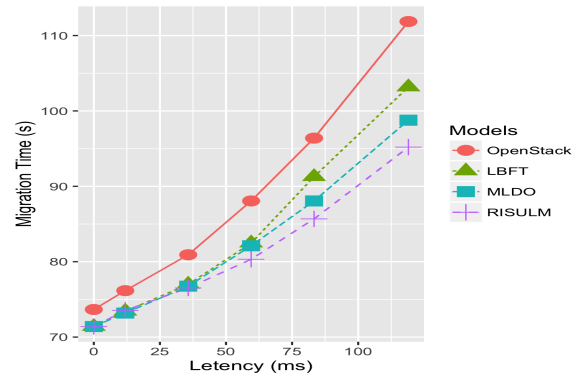
Figure 6.25 Pre-Copy KVM Models.
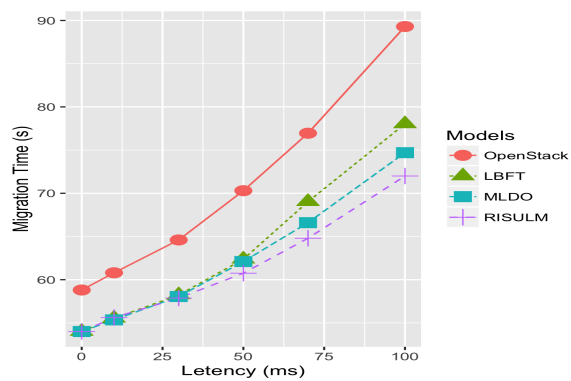

Figure 6.27 Pre-Copy Xen Models.

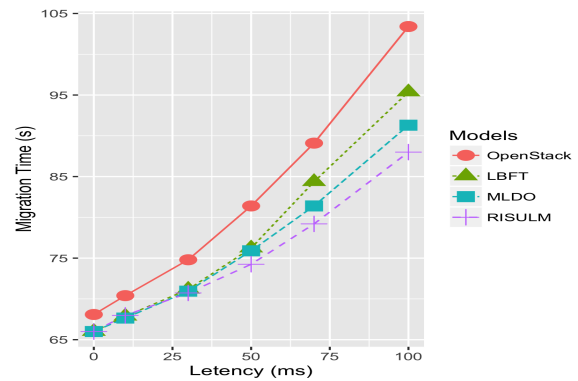Figure 6.26 Post-Copy KVM Models.

Figure 6.28 Post-Copy Xen Models.
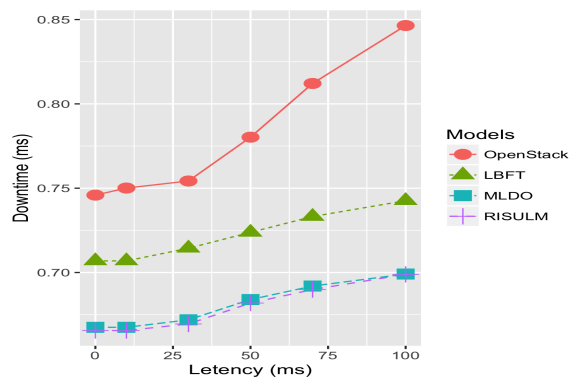
Effect of Latency on Migration time.
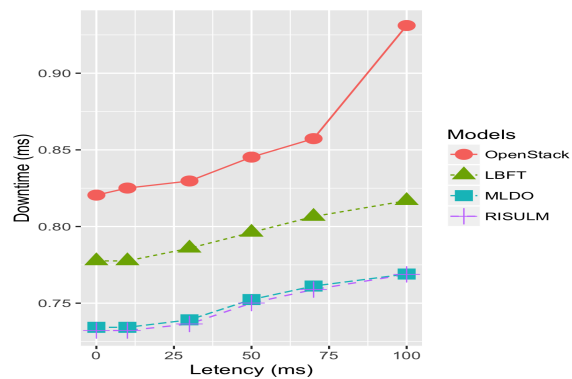
Figure 6.29 Pre-Copy KVM Models.

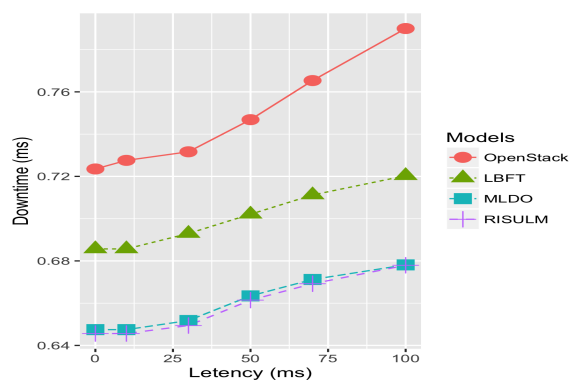Figure 6.31 Pre-Copy Xen Models.

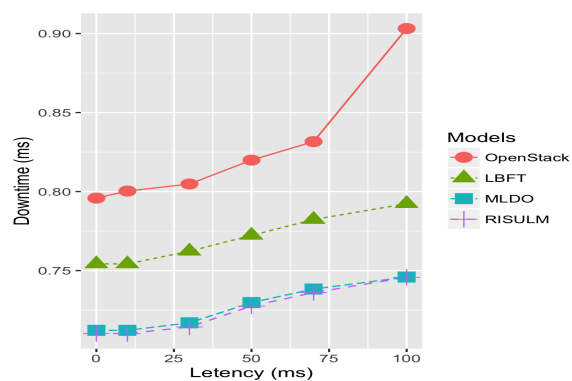Figure 6.30 Post-Copy KVM Models.

Figure 6.32 Post-Copy Xen Models.

Effect of Latency on Downtime.

accuracy on predicting failures than Xen and post-copy algorithm performs better than pre-copy in terms of optimizing migration time and downtime.

This observation is important to note because depending on the type of application being deploy on the VM during live migration, the choice of the algorithm and hypervisor might affects how likely it will success to migrate, which might also affects the migration time and downtime of the VMs.

we observe the following decreases in migration time and downtime respectively as shown in table 6.2, while using different combination of algorithm and hypervisors. Results are summarized as:

Table 6.2 Percentage Reduction on Migration time and Downtime for Different hypervisors and Algorithms.

Model	Migration Time	Downtime	Technology
RISULM	74%	21%	Post-copy KVM
	66%	17%	Pre-copy KVM
	71.2%	19%	Post-copy Xen
	64.5%	15%	Pre-copy Xen
MLDO	62%	13%	Post-copy KVM
	57%	11%	Pre-copy KVM
	60%	11%	Post-copy Xen
	55%	9%	Pre-copy Xen
LBFT	55%	11%	Post-copy KVM
	51%	11%	Pre-copy KVM
	53%	9%	Post-copy Xen
	48.2%	9%	Pre-copy Xen

Moreover, there was no significant different of either downtime, migration time when the workload was below 25% (idle conditions); the VMs have identical characteristics. For these reasons, we sub divided our workloads into 25%, 50% and 75% (idle, average and full loaded conditions) of workloads, before we live migrate the VMs.

We observe that flavor types tiny (t), small (s), and medium (m), we have relatively smaller data transfer rate than the large (l) and xLarge (xl) flavors. Moreover, with flavor m, there is a huge increase of data transfer rate see Figure 6.1, 6.2, 6.3, and 6.4. In Table 5.1 from t to xl, the flavors parameters for Disk and RAM are doubled linearly, i.e., Disk size for t = 10G, s=20G etc., and CPU except for t and s with 1 vCPU each. Our setup and configuration for live migration uses shared storage (NFS), which means the disk could not be any possible reason for this sudden

increase of data transfer rate. Therefore, the RAM and vCPU should be responsible, which justify why we use these parameters for our metric studies.

Furthermore, this hike in data transfer also affects the total migration time per flavor type, see figures: 6.5, 6.6, 6.7, and 6.8 as the migration time grows exponentially irrespective of the algorithm used but differs in magnitude depending on the hypervisor type, for example KVM in general have a lower migration time than Xen and post-copy has a shorter migration time than pre-copy. On the other hand, we observe a different pattern on the downtime see Figures: 6.9, 6.10, 6.11, and 6.12. (t and s), (m and l) having the same amount of downtime, which is contrary to our expectations, we were expecting to see different amount of downtime for t,s and m. Also, we were expecting to see a hike downtime for l and xl flavors due to their parameters, but it didn't turns out to be as expected, the downtime actually grows as a step function. From this observation, we revisit the pre-copy 3.3, and post-copy 3.4 algorithms mentioned above, we noticed that pre-copy has a longer downtime than post-copy. This is consistent with our results as shown above.

RISULM Takes all these observations into consideration, as it monitors the environment of the system (Nodes and VMs) before making a decision to migrate a VM across Nodes.

Another finding we observed is the effect that network links speed and latency has on both migration time and downtime.
On link speed (our switch goes up-to 1GPbs), on all the flavors type, we vary the link speed and measure the migration time and downtime of the VMs instances. We observe the impact that link speed has on both migration time and downtime, and report our results. For migration time against link speed, we report our results in Figures: 6.17, 6.18, 6.19, and 6.20 and for downtime we report our results in Figures: 6.21, 6.22, 6.23, and 6.24.

Regarding latency, we measure the time it takes to transfer data on the VMs during live migration, as the VMs goes from source node to destination node.

We use all four combinations of algorithms and hypervisors in our study, on RISULM, MLDO, and LBFT models including the OpenStack scheduler. We observe the effects latency has on both migration time and downtime, and report the results of our findings for migration time in figures: 6.25, 6.26, 6.27, and 6.28 and for downtime in figures: 6.29, 6.30, 6.31, and 6.32.
Our observations suggest that, migration time increases exponentially as latency increases. Initially, between 0 to 10 ms latency, RISULM, MLDO and LFBT were at par but clearly distinct after 3 ms where RISULM started deviating from MLDO and LBFT throughout whereas MLDO moves at part with LBFT up-to 50 ms. This clearly shows that in most applications, RISULM will be optimal in terms of latency. For downtime, we had these observations: As the link speed increases, we

observe a shape linear drop of downtime between 13ms to 25 ms, (we had observe an exponential decrease of migration time within this range earlier). and from 25 ms to 1Gbps, the conclusion we make here is that even though both migration time and downtime are directly proportional, their relationship is not necessarily linearly.

The reported p-value and cliff's delta values in tables 5.4, 5.5, and 5.6 show statistically significantly p-values, which implies that, the p-values are significantly different with RISULM showing better performance than MDLO and LBFT models.

Therefore, based on the results of our findings, we can conclude that, RISULM outperforms both MLDO and LBFT modes in terms of accuracy, migration time and downtime.

RISULM was also tested for horizontal scalability and robustness using the fault injection framework. Also, our approach is capable of predicting where and when VMs should be live migrated to.

Since we have implemented a prediction model, we observe both categories of error, which are involved in prediction; that is false positives and false negatives. Moreover, RISULM on average tolerates false negatives more than it will tolerate false positives see Table 6.1, this is desirable for the kind of problems we are anticipating to solve, which might heavily penalize false positives than it would for false negatives, because, if our model should predict migration as successful whereas it fails, this is a situation we don't want (no tolerant to this kind of error).

6.4 Threats to Validity

In this section, we discuss the limitations of our work following common guidelines for empirical studies.

- **Construct validity threats** Relates to the meaningfulness of our measurement results, in other words, this threat is solely due to errors in the measurement of metrics. To compute our metrics, we use top, $htop$, $iftop$ and other scripts to capture parameters for our studies such as RAM usage, CPU, Bandwidth, VIRT, dirty page rate, etc. We did not measure the accuracy of these tools and their accuracy can have direct impact on our results. However, they have been used in the literature repeatedly. Additionally, there is a latency when reading data from the log files and our data structure on the shared location (NFS); therefore values obtained might be staled sometimes. In the future, we plan to learn resource usage patterns to overcome this issue.

- **Threats to internal validity** Relates to alternative explanations to describe our finding. The implementation of the MDP and ML-Model can be a threat to our study, moreover, our design window (the time frame we set to capture environment parameters) can also be a potential threat because all the Nodes are not running the same application synchronously, even though the time is synchronized. Also, in this thesis we didn't considered synchronizations between the controller Node and its replica(s), which could have an impact on the observed migration time and downtime. We plan to investigate this in the future.

- **Threats to External validity** Relates to possible generalizing our studies. More validation with different machine learning techniques, possible unsupervised techniques. Different method of ensemble could be used as well to understand the impart of MDP on migration time and downtime.

- **Reliability validity threats** Replication is important in science. Hence, we have attempted to provide all necessary details to replicate our study in this thesis.

- **Conclusion validity threats** Describe relation between the treatment and it outcome. To address this, we were attentive not to violate the assumptions of our null hypothesis.

CHAPTER 7   CONCLUSION

7.1   Summary

In this study, we examine live migration of virtual machines in OpenStack cloud. We proposed a novel approach, RISULM that uses supervised and reinforcement learning techniques to study live migration. This approach aims at studying if failures in live migration could be accurately predicted and if the MDP model could improve live migration scheduling in order to accurately determine where and when virtual machines should be migrated. Moreover, we compared the performance of RISULM against state of the art models from the literature, i.e., MLDO and LBFT, and found that RISULM outperforms both models in terms of accuracy, migration time and downtime.

To simulate a real world scenarios, we used a fault injection framework, which we adapted in our implementation to inject faults on targeted Nodes. In other to test the accuracy of our models, we collected real-time data of the system for both models.

Our results suggest that, we can accurately predict failures of VMs with a 95% accuracy. Our results also suggest that MDP model can improve live migration scheduling, reducing the downtime of VMs by approximately 21% and migration time by approximately 74%.

We also implemented our proposed model; RISULM on different hypervisors using both the pre-copy and post-copy algorithm. Results suggest that post-copy algorithms have better migration time and downtime as compare to pre-copy algorithm. Which suggests that depending on the application, the choice of hypervisor and algorithm to use is critically important.

7.2   Limitations of our Approach.

Even though our results are promising, our approach still have some limitations. For example, we did not considered a real world scenario for a natural disaster, which actually disrupts power, communications links, etc., also, our implementations use a central Controller Node, which doesn't fully reflects the distributed nature of the cloud. Moreover, we did not implement concurrent scheduling decisions in our MDP approach, and hence we could not investigate cases of conflicting policies, i.e., multiple VMs being picked for migration to the same destination at the same time. Presently, the training time for our MDP policy scheduler depends on the size of the cluster, which affects the cost (and consequently the frequency) of trainings.

7.3   Future Work

For future work, we would like to address our limitations, first, by implementing replicating Nodes on the Controller and also addressing the problem of concurrent policy scheduling, which will avoid conflicting scheduling policies. We would also like to improve our MDP training time as the size of our cluster increases. Furthermore, we would like to simulate a real world scenario of a natural disaster, where power is disrupted on some nodes and communications links are interrupted between some nodes, to assess the robustness of our proposed approach.

Then, we would like to suggest our model to be implemented in OpenStack cloud or any other cloud infrastructure, we will also like to extend this study using many Nodes in different geolocations, to investigate if we could obtain the same promising results. In addition, we would like to implement RISULM on more hypervisors such as VMWare and Hper-v etc., and experiment with real world used cases.

REFERENCES

Akoush, Sherif and Sohan, Ripduman and Rice, Andrew and Moore, Andrew W. and Hopper, Andy (2010). Predicting the performance of virtual machine migration. *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE Computer Society, Washington, DC, USA, MASCOTS '10, 37–46.

Arif, Moiz and Kiani, Adnan K. and Qadir, Junaid (2016). Machine learning based optimized live virtual machine migration over wan links. *Telecommunication Systems*, 1–13.

Armbrust, Michael and Fox, Armando and Griffith, Rean and Joseph, Anthony D. and Katz, Randy and Konwinski, Andy and Lee, Gunho and Patterson, David and Rabkin, Ariel and Stoica, Ion and Zaharia, Matei (2010). A view of cloud computing. *Commun. ACM*, *53*(4), 50–58.

V. J. D. Barayuga and W. E. S. Yu (2015). Packet level tcp performance of nat44, nat64 and ipv6 using iperf in the context of ipv6 migration. *IT Convergence and Security (ICITCS), 2015 5th International Conference on*. 1–3.

E. Barrett and E. Howley and J. Duggan (2011). A learning architecture for scheduling workflow applications in the cloud. *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*. 83–90.

Bell, Tim and Lane, Ryan and Martin, JC (2013). Openstack user survey statistics. `http://www.openstack.org/blog/2013/11/openstack-user-survey-statistics-november-2013/`.

M. I. Biswas and G. Parr and S. McClean and P. Morrow and B. Scotney (2016). An analysis of live migration in openstack using high speed optical network. *2016 SAI Computing Conference (SAI)*. 1267–1272.

Blaser, Rico and Fryzlewicz, Piotr (2016). Random rotation ensembles. *J. Mach. Learn. Res.*, *17*(1), 126–151.

Lars Buitinck and Gilles Louppe and Mathieu Blondel and Fabian Pedregosa and Andreas Mueller and Olivier Grisel and Vlad Niculae and Peter Prettenhofer and Alexandre Gramfort and Jaques Grobler and Robert Layton and Jake VanderPlas and Arnaud Joly and Brian Holt and Gaël Varoquaux (2013). API design for machine learning software: experiences from the scikit-learn project. *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.

Bunyakitanon, Monchai and Peng, Mengyuan (2014). Performance measurement of live migration algorithms. *Msc. Thesis, School of Computing, Blekinge Institute of Technology, 371 79, Karlskrona, Sweden.*

Caruana, Rich and Niculescu-Mizil, Alexandru (2006). An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd International Conference on Machine Learning.* ACM, New York, NY, USA, ICML '06, 161–168.

A. Casanovas and J. Alonso and J. Torres and A. Andrzejak (2009). Work in progress: Building a distributed generic stress tool for server performance and behavior analysis. *2009 Fifth International Conference on Autonomic and Autonomous Systems.* 342–345.

Clark, Christopher and Fraser, Keir and Hand, Steven and Hansen, Jacob Gorm and Jul, Eric and Limpach, Christian and Pratt, Ian and Warfield, Andrew (2005). Live migration of virtual machines. *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2.* USENIX Association, Berkeley, CA, USA, NSDI'05, 273–286.

De Poalo, Tracy and Howard, Jeremy (2014). Predictive modeling in practice: A case study from sprint. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, New York, NY, USA, KDD '14, 1517–1517.

Erl, Thomas and Puttini, Ricardo and Mahmood, Zaigham (2013). *Cloud Computing: Concepts, Technology & Architecture.* Prentice Hall Press, Upper Saddle River, NJ, USA, première édition.

Gustafsson, Erik (2013). Optimizing total migration time in virtual machine live migration. *Msc. Thesis, Institutionen för informationsteknologi, Department of Information Technology, Uppsala University, Sweden.*

S. Ismaeel and A. Miri and D. Chourishi and S. M. R. Dibaj (2015). Open source cloud management platforms: A review. *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing.* 470–475.

P. Jamshidi and A. Ahmad and C. Pahl (2013). Cloud migration research: A systematic review. *IEEE Transactions on Cloud Computing*, *1*(2), 142–157.

S. Kikuchi and Y. Matsumoto (2011). Performance modeling of concurrent live migration operations in cloud computing systems using prism probabilistic model checker. *2011 IEEE 4th International Conference on Cloud Computing.* 49–56.

Li, Ziyu and Wu, Gang (2016). Optimizing vm live migration strategy based on migration time cost modeling. *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*. ACM, New York, NY, USA, ANCS '16, 99–109.

Liaqat, Misbah and Ninoriya, Shalini and Shuja, Junaid and Ahmad, Raja Wasim and Gani, Abdullah (2016). Virtual machine migration enabled cloud resource management: A challenging task. *arXiv preprint arXiv:1601.03854*.

Libvirt, Documentation (2016). Virtual machine live cycle of virtual machine, showing various states. `http://wiki.libvirt.org/page/VM_lifecycle`.

Macbeth, Guillermo and Razumiejczyk, Eugenia and Ledesma, Rubén Daniel (2011). Cliff's delta calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, *10*(2), 545–555.

I. E. A. Mansour and K. Cooper and H. Bouchachia (2016). Effective live cloud migration. *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. 334–339.

Mukaka, MM (2012). A guide to appropriate use of correlation coefficient in medical research. *Malawi Medical Journal*, *24*(3), 69–71.

J. P. Mullerikkal and Y. Sastri (2015). A comparative study of openstack and cloudstack. *2015 Fifth International Conference on Advances in Computing and Communications (ICACC)*. 81–84.

Nathan, Senthil and Bellur, Umesh and Kulkarni, Purushottam (2015). Towards a comprehensive performance model of virtual machine live migration. *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, New York, NY, USA, SoCC '15, 288–301.

OpenStack (2016). Openstack survey statistics report. `https://www.openstack.org/assets/survey/April-2016-User-Survey-Report.pdf`.

Qian, Ling and Luo, Zhiguo and Du, Yujian and Guo, Leitao (2009). *Cloud Computing: An Overview*, Springer Berlin Heidelberg, Berlin, Heidelberg. 626–631.

Salfner, Felix and Tröger, P and Richly, M (2012). Dependable estimation of downtime for virtual machine live migration. *International Journal On Advances in Systems and Measurements*, *5*(1).

J. Wang and X. Xiao and J. Wang and K. Lu and X. Deng and A. A. Gumaste (2016). When group-buying meets cloud computing. *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 1–9.

Y. Xiaoyong and L. Ying and W. Zhonghai and L. Tiancheng (2014). Dependability analysis on open stack iaas cloud: Bug anaysis and fault injection. *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*. 18–25.

Xiong, Caiming and Johnson, David and Xu, Ran and Corso, Jason J. (2012). Random forests for metric learning with implicit pairwise position dependence. *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, KDD '12, 958–966.