Workload Assignment in various Heterogeneous Cloud Environments

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

MS in Computer Science and Engineering
by
Research

by

Vishrut Mehta
201102128
vishrut.mehta@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
May 2018

Copyright c Vishrut Mehta, 2017
All Rights Reserved

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Workload Assignment in various Heterogeneous Cloud Environments" by Vishrut Mehta, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. Vasudeva Varma

To Spontaneity

Acknowledgments

Thank you, Prof. Vasudeva Varma for being my advisor, and motivating me to pursue research in a really passionate manner. Your suggestions have given me direction when I most needed it. Our discussions have inspired me to explore the wonderful world of search and information extraction.

I thank Prof. Reddy Raja for all the constructive feedback, and guidance. There was always the spirit of curiosity and involvement during your lectures. I thank all the members of Search and Information Extraction Lab for being there whenever I needed any advice. Your support has been most valuable in this work.

My heartfelt thanks to Tushant Jha for carrying out elaborate discussions and helping me with some theoretical concepts. Thanks to Shashank Sahni, Dharmesh Kakadia, Pulkit Goel for their suggestions at various point of time during my research.

I thank all my friends who have shared the interest towards system and cloud research and helping me out for the same. I'd like to thank all the amazing people who've proofread the drafts and made many useful suggestions – Kumar Rishabh, Himamshu Sharma, Vibhav Srivastava and Meenal Goyal.

From the bottom of my heart, Thank my mom and dad, for your love, support and advice. Thank you for teaching me to be inquisitive, and amazed at the magnitude of this universe.

Abstract

More than a decade ago, startups, companies, research groups or individuals had to purchase hardware and set it up for each task. This approach made sense when the utility and the requirement of the hardware were fixed. However, this does not make sense in today's growing market, since there is a continuous increase or decrease in hardware's demands. Often, people require large amounts of resources for short duration of time (like in the case of scientific computation). In other cases, people require the ability to scale up and down the number of systems based on the demand rendering dedicated systems unprofitable. Resources can be of different types such as computation(CPU), storage(distributed file system storage), network, databases, etc.

Cloud computing opens up a whole new paradigm where individuals or companies can rent all the resources they need and dispose them off later after the work is done. In this process, they are charged for exactly what they use and not a penny more. This is therefore a highly flexible model, since one can add or reduce the resources on the fly according to the requirements and be charged accordingly. With data centers spread across the globe and the use of cloud computing, we can now localize resources closer to the end user to improve performance and experience. Cloud Computing is thus highly profitable as it is based on utilization of commodity hardware. The systems are heterogeneous in nature as they have different memory performance, different storage capabilities, different network capabilities etc. In today's world, we cannot have homogeneous systems as they will not be able to provide the flexibility that we require. For example, if some nodes in the cluster are over-utilized and some disks in the storage cluster are under-utilized, there can be a lot of scenarios where homogeneous systems won't work due to the dynamic nature of our workloads or tasks. We therefore need to develop systems that are cognizant of this inherent heterogeneity and can adapt to the constantly changing environment.

In our work, we are focused on allocating resources for workloads which are short lived and mainly used for batch processing. During a computing process, workload is the amount of processing that the computer has been given to do at a given time. So our main focus is to allocate resources from a specific cloud provider for handling these dynamic workloads. These assignments could be on the basis of price, utilization, individual cloud characteristics, efficiency/performance etc. We try to use all these factors to design various techniques for the assignment of different types of heterogeneous scenarios. Below are

the three scenarios of heterogeneous cloud systems for which we have designed frameworks/algorithms to assign workloads:

In the first scenario, we propose MultiStack, which is a big data orchestration platform for deploying big data jobs across the hybrid cloud environment. The specific architecture elaborated in this paper uses Amazon Web Services as the public cloud provider and Openstack as the private cloud framework. The proposed framework aims at reducing the job completion time of workloads along with decreasing the cost using Spot Instance provisioning instead of on-demand provisioning. In this case, we try to assign an optimal number of instances (cluster of computing resources) to the given workload along with cost minimization.

In the second scenario, we propose a solution for optimal IO Workload assignment with respect to backend storage (Software Defined Storage) using statistical modelling to estimate measures of performance such as Throughput, IOPS, et al in that storage system. The proposed system uses support vector regression to estimate the performance of individual IO Workloads on each available Software Defined Storage(SDS) system for optimal assignment. We demonstrate our solution in a heterogeneous environment comprising of HDFS, GlusterFS, and Ceph. Therefore, in this case, we try to increase the efficiency by choosing/assigning an optimal storage environment for any given type of workload.

In the third scenario, we try to provide optimal assignment of IO workload with respect to compute and storage for a multi-cloud heterogeneous systems. We try to achieve and represent "optimality" by designing a pipeline model which selects a compute and a storage class for a given workload to maximize efficiency and then apply discount schemes provided by the public cloud providers for cost minimization. We also use some input parameters provided by the user to fine tune the cost minimization procedure. We have designed these methods and algorithms using the following three public cloud providers: Amazon Web Services, Microsoft Azure and Google Cloud Platform.

We expand on these different scenarios and try to build different methods to solve each of them. Through this work, we hope to help individuals, startups, research groups or even companies in making an informed decision about choosing a cloud provider vs using their own local setup, choosing a storage type for their needs or comparing different cloud provider for a particular type workload and thus use these frameworks to benchmarks their resource utilization.

Contents

List of Figures

List of Tables

Related Publications

Mehta Vishrut, Kumar Rishabh, Reddy Raja, and Vasudeva Varma. "MultiStack: Multi-Cloud Big Data Research Framework/Platform." In Cloud Computing in Emerging Markets (CCEM), 2016 IEEE International Conference on, pp. 147-152. IEEE, 2016.

Kumar Rishabh, Vishrut Mehta, Tushant Jha, and Vasudeva Varma[1]"HetStore: A Platform for IO Workload assignment in a heterogeneous storage environment." In Cloud Computing in Emerging Markets (CCEM), 2016 IEEE International Conference on, pp. 25-31. IEEE, 2016. HetStore: A Platform for IO Workload assignment in a heterogeneous storage environment. In Cloud Computing in Emerging Markets (CCEM), 2016 IEEE International Conference.

Vishrut Mehta and Vasudeva Varma. CM(EM): Efficiency Maximization and Cost Minimization for IO Workload assignment in a Heterogeneous and Multi-Cloud Environment.
(Under Review)

[1]The first two authors have equal contribution.

Chapter 1

Introduction

Cloud computing provides on demand shared computer processing resources and data to computers and other devices and serves as an Internet-based computing solution. We try to leverage all the computation flexibility it offers in terms of network, compute, storage, etc.

1.1 Cloud Computing

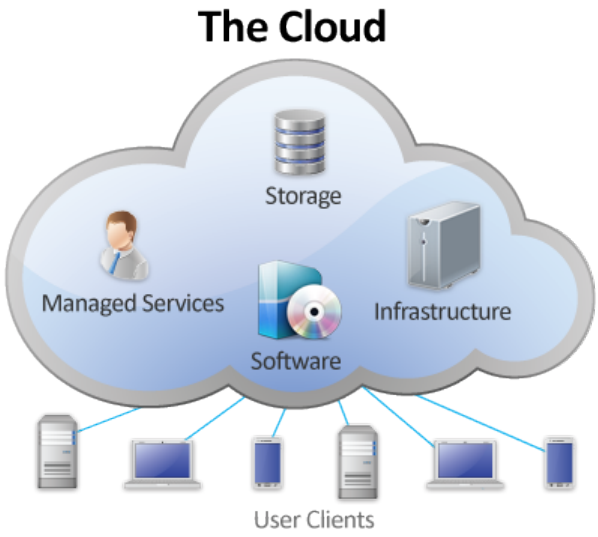

Figure 1.1: The Cloud

A remote shared pool of resources can be rented out by organizations for their use rather than using their own resources with the advent of Cloud Computing. There are many advantages of such a system. First, you pay only for the resources that you use which makes it is highly economical for small-scale purposes. Second, it is on-demand and can be easily expanded as per the requirements of the user, helping the application to scale easily. Third, reliability is offered by cloud service providers since they have data centers in multiple sites and can easily transfer the load to another data center in case of any problems. Finally, maintenance of the resources is in the hands of the cloud service provider and the user doesn't need to worry about it.

We have the following three cloud scenarios possible, namely, public, private and hybrid:

Public Cloud

In a public cloud, companies or organization host their own resources (network, memory and storage) for all users by using "pay for what you use" model. This system is helpful to overcome the extra cost overhead of under utilized resources. The advantages of this solution is that the customer can deploy and test their products quickly but would not control the security.

Private Cloud

A private cloud is a secure cloud environment consisting of the organizations' own compute, memory and network resources. These resources are them accumulated in a complex architecture and allocation of these resources depending on various factors of the workload is decided. The advantages of this solution is that the user will get more control over secuirty and privacy but would have to setup their own infrastructure with their own resources and then creating a virtualized system on top of this.

Hybrid Cloud

This system is made using both the public and private cloud architectures. So a private infrastructure is bound together with a third party cloud provider in which the user could explicitly tell which one to use, or could defines rules (resource allocation algorithms) to efficiently allocate compute, memory and network resources for their jobs. By this, a single system could reap benefits of both the private and public cloud infrastructures.

1.2 Types of Heterogeneous Environments

In our work, we try to focus on virtual heterogeneous and storage heterogeneous environments. We explain the characteristics of these heterogeneous systems in this section.

Heterogeneity in Compute

The key feature used in a heterogeneous cloud environment is virtualization done by the hypervisor. This is responsible for proper allocation of compute resources to the users and provides flexibility like auto-scaling, handling spikes etc.

Heterogeneity in Storage

A heterogeneous storage environment is an amalgamation of various different types of storage systems and backend to read and write data from. This can serve various types of workload with proper storage allocation.

1.3 Types of workloads

"The average amount of work performed by a set of computing resources (network, memory and storage) in a given period of time gives an estimate of the performance and efficiency of that workload." The different types of workloads are:

Memory Workload

Memory workload deals with measuring and analyzing memory usage of the system at a certain amount of time. There could be instances or spikes in memory due to large number of jobs being executed at a certain time demanding more memory to the workload.

CPU Workload

CPU workload is basically related to performance improvements proportional to the number of instructions executed by the CPU. For cost and energy efficiency, the system needs to upgrade or downgrade the processing power (CPU usage) at a given time.

I/O Workload

The input-output workloads of any application deals with the read and write operations of data in a local (in-memory) or a third party storage (public storage) at a certain period of time. It is the responsibility of the system engineers to analyze the load performance parameters required to maintain their system in a stable as well as a flexible state.

Database Workload

Database workload interacts with backend structured data measured in terms of number of queries (like lookup, insert, delete, etc) executed in a certain amount of time. Database performance is approximately measured by analyzing the memory, I/O operation and throughput of the database.

1.4 Problem Definition and Organisation of thesis

In this section, we give a brief summary of the problem we have addressed and tried to solve. Our work mainly involves experimenting various assignment procedures for workload assignment in three different heterogeneous environments. We will give a high level overview of the problem definition and the heterogeneous environment details in this section.

	Type of Heterogeneity	**High-level procedure used**	**Performance Measurement**
Case 1	Compute	cost function and filtering through user inputs	Cost Minimization
Case 2	Storage	statistical modelling	Efficiency Maximization
Case 3	Both of the above	Both of the above	Both of the above

Table 1.1: Summary of the problem definition

In Chapter 2, we tackle a hybrid cloud scenario where we have a set of public cloud systems (AWS, Google Compute or Microsoft Azure) and a private cloud provider (Openstack). We schedule batch processing map reduce workloads on this hybrid environment. The problem that we try to solve in such a hybrid scenario is to identify which particular type workload will be allocated to the public cloud system and which types will be allocated to the private cloud system. We also determine how many instances/resources would be needed to execute this workload. We then allocate resources to the workload while minimizing the cost incurred to run this workload.

In Chapter 3, we take a heterogeneous scenario of different software storages and try to allocate our workload in any one of these software defined storages. Software Defined Storage frameworks typically decouple the task of managing and provisioning data from the underlying hardware. We predict the performance measurements and simulate a training data to evaluate what type of workloads performs better with respect to efficiency (cost function) and also in which software defined storage solution. We then maximize the efficiency of executing our workload by choosing the optimal SDS solution for a particular given workload.

In Chapter 4, we have a multi-cloud scenario, where we take three well-known public cloud providers - Google Cloud Platform, Amazon Web Services and Microsoft Azure and try to assign a particular given IO workload to one of these public cloud provider. We take into account all the

discount schemes the cloud provider has to offer and also the flexibility in their heterogeneous environment itself. We assign particular compute and storage resources to the workload such that we have the maximum efficiency and the minimum cost for a given workload. We address this trade off and show some comparison for these workloads on these three cloud providers.

In the last chapter, we conclude our thesis with the hope that these experiments would help researchers and people in the industry in choosing the right computing resource for their custom workload. In the end, we present this thesis with some open thoughts and future ideas to extend our work and help the industry gain the maximum from the computing power it possesses.

Chapter 2

Workload Assignment in a Heterogeneous Virtualized Environment

2.1 Introduction

Due to virtualization, we can now achieve higher flexibility into the system as well as take several measures to reduce cost and increase performance. Many organization would setup their in-house setup to efficiently use their resources, called private clouds. These setup were costly but gave greater control over the system and security for their data. But the era that public clouds brought was quite remarkable. Now you can buy a resource and would be billed only for the hours/minutes you use it. This type of flexibility was observed in various types of network, compute and storage resources with greater security and privacy measures. Everything was running and stored in the so-called "cloud". Both these approaches have their own advantages and disadvantages, so a new system combining the power of both public and private cloud was born - "hybrid cloud".

What is hybrid cloud ?

The amalgamation of both service providers - public and private cloud into a single integrated architecture (used by an organization) to provide a compatible environment to interact with each other over an encrypted channel. The popular public cloud providers are Amazon Web Services[1], Google Compute Engine[12], Microsoft Azure[17], etc and private clouds such as Openstack[19], CloudStack[7], etc.

Why hybrid cloud ?

Flexibility

There is tremendous flexibility in terms of running workloads for the best suited cloud provider considering the both in-house cost vs lease cost giving greater efficiency and performance. A whole application which has multiple components like database, disk storage, web servers, etc

could run on any of the cloud provider which meets their particular needs like for a database server or disk, we would need higher I/O operations. So this enables us to classify the resources to the workloads' needs and use them accordingly based on several user defined heuristics.

Control

Custom permissions or groups can be set in this system giving higher control for the use or the demand of resource. Major control of the system is achieved by setting custom network configurations. The network could be segmented wherever needed. Also, there is always an option to setup and upgrade your own resources for specific requirements.

Security

In hybrid scenario, both cloud provider i.e. the dedicated servers and the public resources could interact on private network eventually becoming single system.

Billing

There are batch processing jobs which you need to run quickly which would give us the ability to auto-scale without manual intervention, but there are some workload where you need to run on dedicated resources for meeting their needs of performance. The combined cost of using the mixture of these solutions is significantly reduced.

2.1.1 Scheduling Techniques

While focusing on hybrid scenarios, the scheduler will be benefited greatly by:

Deadline awareness: We need to know deadlines to make effective bidding and scaling for big data clusters.

Cost awareness: Due to increasing demand in compute resource for large data processing jobs, people are migrating to online cloud solution for these data processing needs. This increases the cost of their work and need to track and optimize this added incurred cost.

2.1.2 Multi-Cloud

"Multi-cloud is the use of multiple cloud computing services in a single heterogeneous architecture." For example, a single organization could simultaneously use different cloud providers at the same time to fulfill their requirements.

Services

In this section we identify and discuss services that a multi-cloud platform should provide.

Resource Management

Resource management is one of the core problem for computer science. It affects the three basic criteria for system evaluation: performance, functionality and cost. In failure of doing so, it may significantly impact the performance and cost. In a multi-cloud environment, the platform manages not only for traditional resources like CPU, memory and storage, but also responsible for resources like cost, IP address allocation etc.

Monitoring Traffic and Data Management

Due to multiple service providers working together, the cumulative monitoring of traffic for all resources becomes a major necessity. Also, data management plays a major role to distribute data strategically across different data centers to reap benefits of geographhical locations.

Billing and Quota Enforcement

A unified quote thresholds are required to maintain their financial structure. An allocation strategy is required to divide the resources to user or their groups with respect to different available cloud providers.

Authorization and Authentication

In this new cloud federation, the main difficulty is to maintain the identity and authentication between different clouds.

2.2 Related Work

Apache Hadoop [4] has been an enabler for big data processing on commodity hardware. The adoption of cloud has made it possible to run hadoop cluster of very large size within minutes. Although hadoop has been a phenomenal success, configuring and managing hadoop clusters has been non-trivial. As organizations are moving to virtualized environment, the need to run hadoop seamlessly and effectively over virtual machine clusters is emerging. The possibility of using both public and private cloud for running jobs respecting deadlines and minimizing the cost is exciting. As early adopters of both hadoop and cloud, we believe resource optimization across multiple cloud is one of the most important issue. Over years, we have tried to improve our infrastructure for better management and utilization of resources. We shifted a majority of our systems to private cloud using OpenStack. We are now experimenting with the possibility of optimizing and automating hadoop deployments on cloud. We propose

MultiStack, a system for running on-demand hadoop across multiple cloud.

Existing Solutions/Tools

In this section we would like to review existing tools with similar aims.

VMWare Serengeti

The open source project Serengeti [21] is an initiative by VMWare to bring virtualization benefits to hadoop. Its limitation is, the full support is available mainly to VMware vSphere based virtualization. Although, other platform are supported but with limited features.

OpenStack Savanna

This is an open source effort mainly led by Mirantis inc. [1] It is mainly limited to OpenStack and Hadoop.

Amazon Elastic MapReduce

Amazon provides a Elastic MapReduce Service [3] which has overlapping goals. Allows to use spot instances to scale. It is limited only to Amazon.

Hadoop on Azure

To summarize, current state of art addresses subproblems of our goal. Combining them is not possible due to many issues like licensing, different architectures, etc.

2.3 MultiStack

MultiStack is a platform for multi-cloud research which simplifies multi-cloud research by providing common functionalities, like multi-cloud OS. Big data is the best use case for cloud computing.

We propose to build a middleware for Hadoop family of project for on demand provisioning and smart scheduling of hadoop workload/jobs, MultiStack [13].

MultiStack will provide users the ability to run different Hadoop jobs on different cloud providers. It provides both command line and web-based clients to submit jobs. It employs smart scheduling to estimate job completion time and costs involved, and schedule them accordingly to minimize both.

We have designed a system bringing previously discussed concepts to real world application.

MultiStack brings together the ability to process big data using various frameworks onto various public and private clouds. In the current implementation, MultiStack uses Amazon EC2 API [2].

[1] http://www.mirantis.com/

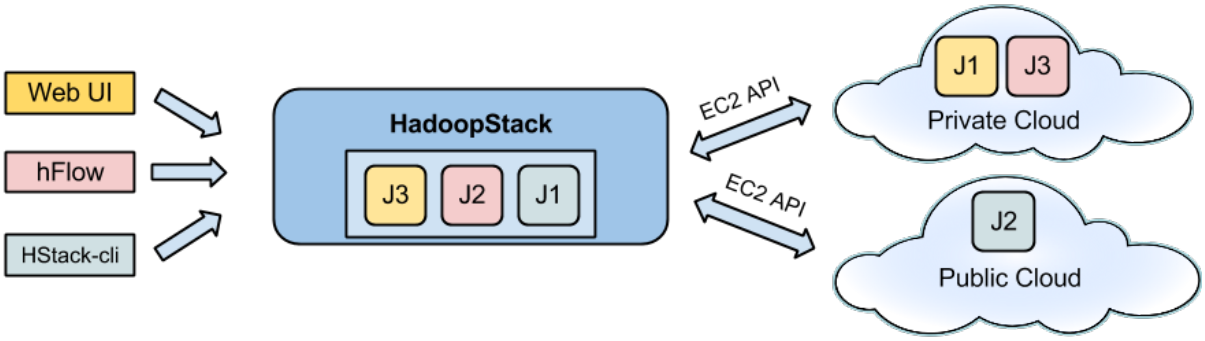

Figure 2.1: MultiStack Overview

2.3.1 Cost-aware Scheduling

Spot Instances

AWS(Amazon Web Services) has a marketplace to allow users to bid on the spare EC2 resource called the spot instance. They are available on highly dscounted rates compared to the normal instances i.e. On-Demand instances. With these instances you can increase resources (compute power/storage capacity) for your application at a very low cost. The main differences between Spot instances and On-Demand instances are that Spot instances might not start immediately, the hourly price for Spot instances varies based on demand, and Amazon EC2 can terminate an individual.

$$OptimalSpotinstanceCount = \arg\max_{i}((d_m \quad d_i) \quad (Cost(i) \quad d_i) \quad (\quad i)) \qquad (2.1)$$

where,
d_m is estimated deadline with running current set of instances
d_i is estimated deadline after scaling the cluster with i spot instances
$Cost(i)$ is estimated cost of running i spot instances. This is simple regression over past pricing data.
and represent the weights that controls penalty.

In this equation we are trying to maximize the gain achieved by adding i spot instances. The first term captures time saving by adding extra instances. Second term represents the cost of running i spot instances. We penalize spawning many more instances by using third term.

In Algorithm 1 we describe scaling algorithm with spot instance. In [25], the author proposes the use of spot instances to accelerate map reduce jobs. Authors also note the challenges that arise due to transit nature of spot instances.

Algorithm 1 Cost-aware autoscaling of clusters

1: Monitoring data about each Job.
2: Find optimal type of instance for scaling.
3: Predict the expected finish time with current set of instances.
4: **if** Current spot instance cost < Upper limit by user **then**
5: Find optimal number of spot instances to add using spot instance price history and deadline prediction using equation 2.1.
6: **end if**
7: return *instance_count*

2.3.2 Deadline-aware Scheduling

In many enterprise, data processing jobs are quite repetitive. James Horey *et al.* [28] presents challenges and approaches for providing big data as a service. A. Ganapathi *et al.* [26] presents statistical machine learning based methods for predicting job completion time as function of provided resources. This scheduling scheme is dependent on the factor that a certain job need to be completed in a given timeline.

2.3.3 Policy Based Scheduling

The large scale enterprises are guided by policy decision reflecting business needs. Traditionally the policy dictated a variety of decisions mainly controlling resource allocation to different divisions (allocating 70% resources to online services and 30% resources to the batch processing). We try to categorize these policies to incorporate the user and the admin needs. They are:

Pro-active Approach

Pro-active is based on metrics defined by the respective users/admins.

The metrics defined by admin that are currently supported include:

- Storage/Bandwidth Quota per tenant
- Number of job instances per tenant

The metrics defined by users that are currently supported include:

- Job Characteristics (Memory/IO/Storage)
- Cost Limitations per Job

The pro-active approach is supplied in job file in json format. MultiStack will take these parameters and convert them to policies. Before execution of the user-defined, it will do a sanity check of

all the policies defined to see if they are not violated. If they violated, Multistack will ask the user to modify the respective parameters till the policy is honored, else the job would be terminated. For instance, if the cost per job is getting violated, the platform will ask the user to update the job parameters and submit the job again.

The platform boots the EC2 instances mentioned in Table 2.1 based on the job characteristics:

Job characteristic	**EC2 instance type**	**Rationale**
CPU intensive	c3, c4, x1	c3, c4 and x1 instances are compute and memory optimized
Storage/disk IO intensive	i2, r3	i2 and r3 instances are optimized for storage performance and high IO tasks
GPU intensive	g2	g2 instances are optimized for gpu compute applications
General Purpose	t2, m4, m3	t2, m4 and m3 are low cost general purpose instance type

Table 2.1: EC2 Instance type selection on basis of job characteristics.

Reactive Approach

Currently, the Multistack platform tries to take reactive approach to spot instances. To enable the platform to respond reactively, the user must submit cost bid for the spot instances to be used to run the job. Suppose the current cost of spot instance exceeds the user given value, the platform will rerun the job later when the price bid falls below the user-defined price.

2.4 Architecture

Figure 2.2 shows various components and their interactions. The client interactions with the system is through REST calls.

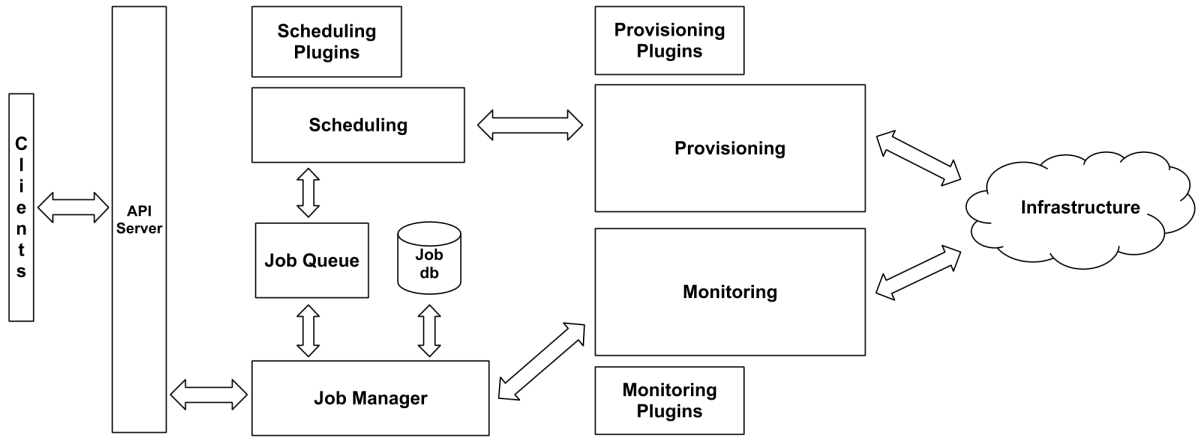

Figure 2.2: MultiStack Architecture

2.4.1 Client tools

Web Interface - The web component of MultiStack will allow a user to submit jobs, mention the input and output location and the deadline. MultiStack accepts EC2 credentials and hence works with most of the cloud providers - AWS, OpenStack, CloudStack, Eucalyptus, HPCloud etc. The resource allocation will be completely transparent to the user and optimized to minimize cost and job completion times.

hstack-cli - This component is built keeping devops in mind. Using command line tool, a variety of jobs can be automated.

hFlow - Its a web interface for composing map-reduce workflows. It allows users to chain different map-reduce functions. These functions can be submitted by a user or may belong to an existing map-reduce repository. We plan to use Apache Oozie [5] as our Workflow Scheduler.

The end user is exposed with only two APIs in this framework to conduct a workflow or independent multiple jobs. For *Cluster API*, the user will have to submit his requirements through a POST request. The parameters needed includes a json text or file as:

```
{
    jobs: {
        id:1
        RAM:4096(in MB)
```

vcpu:4
storage:10240(in MB)
},
{
id:2
.
.
}
}

It would create a cluster in Openstack, if there are resources left, else it would wait until a job is specified for that cluster and then spot instances are created by the provisioning unit. We could also add or delete machines in a cluster whenever needed. For *Jobs API*, the user will have to specify the information for a job along with the job characteristics and the scheduling unit will then analyse the need of spot instances to be created if the user has demanded for lower job completion time which will be described below. We could also add or delete a machine for a current running hadoop job. The parameter needed includes a json text or file as:

{
Link: <jar url>
Command: <hadoop command with input/output>
Cluster: <cluster id>
Cost limit: <in dollars($)>
Job characteristics: < refer from Table 2.1 >

}

The Table 2.2 shows APIs.

2.4.2 MultiStack Server

In this section we describe:

api-server - It provides a REST API for communicating with the MultiStack.

job manager - Responsible for managing and tracking user jobs.

database - For job management.

API Group	HTTP verb	Description
Cluster	POST	Create a cluster
Cluster	PUT	Add machines to existing cluster
Cluster	GET	List cluster(s)
Cluster	DELETE	Destroy a cluster
Job	POST	Run a job
Job	PUT	Add machines to existing job
Job	GET	List job(s)
Job	DELETE	Cancel a job

Table 2.2: MultiStack API Description

scheduling unit - Its a pluggable scheduler interface for scheduling decision of jobs.

provisioning unit - Unit responsible for spawning and configuring instances.

monitoring unit - Aggregates system info from multiple instances.

When the user submits a job request, the Job Manager daemon starts and stores the information in the Job DB with the status of scheduling. Simultaneously, the Scheduling unit is initiated which is the most integral part of the system. It decides the allocation of the resources on the multi-cloud (Openstack and AWS) depending on factors such as resource usage, job completion time, cost limit and the job characteristics specified by the user. As per the given job, MultiStack predicts the job completion time on our private network and then finds an Optimal Spot Instance count i.e. equation 2.1, depending on the spot instance history pricing and deadline prediction after scaling the cluster by i spot instance for a given cost limit specified by the user in the API request. The resource usage statistics are obtained by nova API in Openstack and ec2 API in Amazon Web Services. As the count is decided by the scheduling unit, the job db stores the final resource allocation on the hybrid cloud for this particular job and changes the job status to provisioning. The Provisioning Unit initializes the cluster and adds the instances from Openstack and AWS after the decision of the scheduling unit. After the cluster is initialized, the hadoop jobs are are initialized in the hybrid cloud. This is how Multistack Server works.

One more thing to note that Amazon's Spot Instances have no fixed SLAs. There is a 2 minute warning before any EC2 Spot Instance is terminated. When the spot instance is terminated, the rest of the map reduce jobs left are run in the Openstack cluster until the spot instances are spawned again. When the Spot market price (the price of any spot instance on the EC2 spot bidding portal at any given point) falls back below our bid price, our Spot Auto Scaling group (a service provided by AWS to auto

launch/terminate instance after analyzing their resource requirements) could automatically launch new Spot instances, if needed.

2.5 Evaluation

For Multistack evaluation, we show a tradeoff by reducing the execution time of the job with minimal increase in cost in a hybrid scenario. We will be using Openstack as the private cloud platform and AWS as the public cloud platform.

Flavour	**On Demand Price**	**(Avg)Spot Instance Price**
m1.small	$0.044	$0.0071
m1.medium	$0.087	$0.0082
m1.large	$0.175	$0.0166
m1.xlarge	$0.35	$0.0357

Table 2.3: Amazon Geneal Purpose EC2 Pricing

For testing, we will be using different map reduce programs and would run them on Openstack and AWS individually. We will run a simple wordcount program for different data sizes like 100MB, 500MB, 1GB and 5GB. This example is very standard and its source code is easily available online. We will be creating a cluster of 4 nodes of m1.medium flavour in which we will have hadoop installed. The specification of m1.medium instance constitutes of 2 vcpus, 4GB RAM and 400GB hard disk. For running this testbed/cluster of 4 nodes in AWS, the on demand instance would cost a total of $0.348 per hour as shown in Table 2.3.

From the above Figure 2.3, we could see that compared to Openstack, the average job completion time has reduced by 30-37% and compared to AWS it is still more, but the cost has been reduced from $0.348 to $0.0328. So the average cost has been significantly reduced to 10-30% compared to AWS On-Demand pricing.

2.6 Miscellaneous System Features

Auto Scaling

Based on the deadline, infrastructure and cost requirements, the scheduler will smartly scale the resources allocated to a job. The user is also provided an option to manually scale up/down.

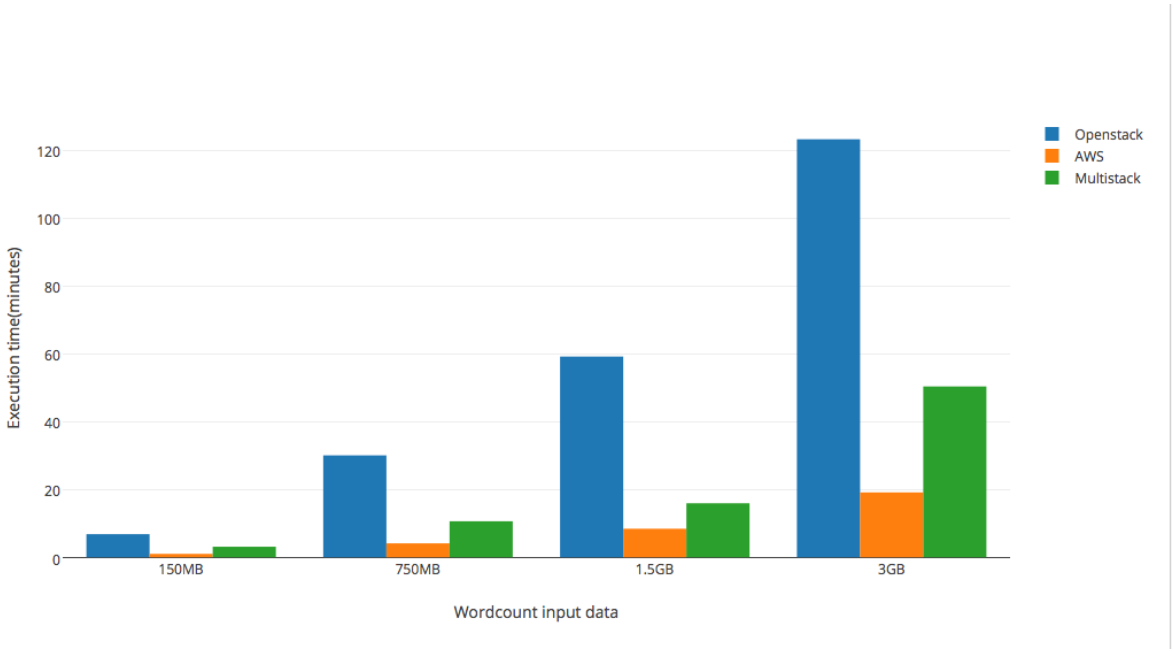

Figure 2.3: Hadoop Job Evaluation

Ability to run across multiple cloud providers

If you have multiple jobs and access to multiple cloud providers, MultiStack provides you the ability to run different jobs on different clouds.

Priority based Job scheduling for minimizing cost and completion time

MultiStack uses machine learning to smartly allocate jobs across multiple cloud providers aiming to reduce your cost and job completion time.

Performance optimization with storage integration

With storage administration privileges, MultiStack auto-replicates the resources that being utilized heavily, thus improving performance.

Client Tools

A web interface and a simple command line interface for interacting with MultiStack.

2.7 Summary

So, in this chapter, we proposed a system - MultiStack, a big data orchestration platform for deploying big data jobs across multiple cloud providers. The specific architecture used Amazon Web Services as the public cloud provider and Openstack as the private cloud framework. Our solution also supports complete Hadoop ecosystem tools - hive, pig, Hbase, oozie etc. and on-demand scaling of Hadoop clusters. Our proposed framework aimed at reducing the Job completion time on workloads along with decrease in cost using Spot Instance provisioning compared to on-demand provisioning by providing two modes of operation: Proactive scheduling and Reactive scheduling, which takes into account user providing job characteristics (eg memory, cpu, etc.), quota limitation and business objectives.

Chapter 3

Workload Assignment in a Heterogeneous Storage Environment

3.1 Introduction

The advent of huge data centers around the world has been marked by gain in prominence of Virtualization Technology. In particular the virtualization of compute/storage has brought access to these resources to end user much cheaper than ever before. Any cloud framework has a set of resources like compute/storage servers that is shared among a number of tenants. The tenants generally interact with the cloud by submitting workloads which require a slice of the compute and storage resources available. All these systems are generally cheap commodity servers which like any distributed system are susceptible to failure. In particular we would want a storage backend to have characteristic features like fault tolerance, replication and deduplication to overcome the underlying hardware failure. A recent trend that is gaining immense popularity to tackle this problem is Software Defined Storage(SDS) [23]. Software Defined Storage frameworks typically decouple the task of managing and provisioning data from the underlying hardware.

There are a number of vendor specific solutions like EMC ViPR[8], NetApp ONTAP[18], IBM Virtual Storage[15] and Opensource Solutions like Ceph[6], GlusterFS[10], HDFS[14], Swift[20] available in the market. Each SDS can be characterised by its strengths and weaknesses owing to the underlying algorithm taking the decisions. For instance, our experiments show Ceph performs better than HDFS when the average file size is small(like a transactional log workload) though HDFS tends to perform better when the average workload size is large[35]. Also the classes of workloads can be pretty diverse. For instance, the workload can be transactional characterised by small files with high iops or photo stream characterised by mid range file size with average iops. A workload can be hybrid of different workload classes too. For instance, a web server may serve photos to users and at the same time have logging daemon running in background.

Hence it makes sense for any cloud vendor to provide user with a heterogeneous storage environment which leverages the strength of various off-the-shelf SDS backends to serve diverse classes of IO workloads. But taking decision where to provision a workload may prove to be a difficult task for a Cloud Administrator. Various approaches have been tried to make the platform workload aware which include policy based provisioning[24], data mining approach[34] and empirical analysis[36]. *To the best of our knowledge this is the first work which takes an analytical approach for classifying workload in heterogeneous SDS environment.*

Using SDS environment instead of traditional storage to provision IO workloads has immense benefit to both users and Cloud Admins. From the Cloud Admin perspective it makes the management of Storage Cloud easier. All the features like replication, fault tolerance and deduplication are available in Software domain and hence does not require any special hardware. This brings down the cost drastically and User can get same premium features at a fraction of cost. The benefits can be gauged by the fact that even a data intensive organization like Cern chose to go with Ceph to build its data storage. It manages the Petabytes of experimental data generated from the most important experiment done by the human race over a Ceph installation[38].

In this chapter, we propose HetStore which is a unified platform for better provisioning of IO workloads over a heterogeneous SDS environment. It does this by using statistical models for classification of an incoming workload to one of the underlying SDS. It has three major components namely the Controller, the SDS cluster and the HetStore Client.

The *Controller* consists of two major components namely the Storage Selection Module (SSM) which models an incoming workload and assigns it to one of the SDS attached. The other module is the Cluster Resource Monitor (CRM) which keeps track of all the metadata pertaining to users and IO Workload and also monitors the running clusters. The *SDS Cluster* is the collection of all the SDS Clusters attached to the platform. The *HetStore Client* consists of light weight daemons that run on each of user machines which are responsible for message parsing and internal communication.

For the purpose of our experiment we have decided to build an SDS environment with Hadoop, Ceph and GlusterFS as SDS backend.

We evaluate our platform in two ways:

Accuracy of statistical regression
After generating the synthetic training set, we evaluate our estimator by cross validating the evaluated training sample (i.e R^2 score denoting the coefficient of Determination[33]).

Comparative efficiency of Assignment
We further demonstrate the accuracy of using these regression models for assigning workloads in order to maximize throughput.

The rest of the chapter is structured as follows - In Section 2, we talk about similar solution pertaining to characterization of IO workloads. In Section 3, we explain the overall HetStore architecture along with the implementations details, such as APIs and parameters. In Section 4, we elaborate on the underlying algorithm of the SSM. In section 5, we validate our prediction accuracy and assign the workload to a cluster. Section 6 concludes the chapter and showcases the vision for the project.

3.2 Related Work

Albrecht *et al.* [24] defines policy based approach for workload partitioning in a tiered cloud dfs between flash and disk storage. Various policy based approaches like LRU, FIFO are used to move workloads in or out of flash tier to disk tier. Seo et al. [34] take data mining route for characterizing IO workload in NAND based SSD environment. They define IO workload classes based on various data mining approaches which could be then exploited by the storage devices such as SSDs.

Kavalanekar *et al.* [31] carry out trace analysis of traces obtained from production servers to develop characterization of traces like block level statistics, multi-parameter distribution etc.

Tremblay *et al.* [36] take empirical approach for workload assignment in heterogeneous SDS environment. They define a workload aware storage platform which takes a set of pre-defined rules for workload assignment in a heterogeneous storage environment.

The gain in prominence SDS backends provides an opportunity to leverage heterogeneous environment to better provision IO Workloads. In this chapter we try to exploit underlying algorithms of each of these backends using statistical analysis to provide a unified solution which performs better than using a homogeneous storage solution on various metrics (like cost, performance, etc) to classify IO Workload.

3.3 Architecture and Implementation

Figure 4.1 shows the high level architecture of HetStore. The *Controller* machine consists of two major components namely the Storage Selection Module (SSM) and Cluster Resource Monitor(CRM).

1. Storage Selection Module (SSM): The SSM module does the modelling of any incoming IO workload and then assigns a particular SDS cluster based on the prediction done by the statistical model generated as described in the SSM Algorithm section. To train this module, we create a synthetic IO workload dataset using the FIO[9] benchmarking tool.

2. Cluster Resource Monitor (CRM): It contains all the information of different SDS clusters attached and monitors the resources allocation of those cluster. It stores the metadata of the work-

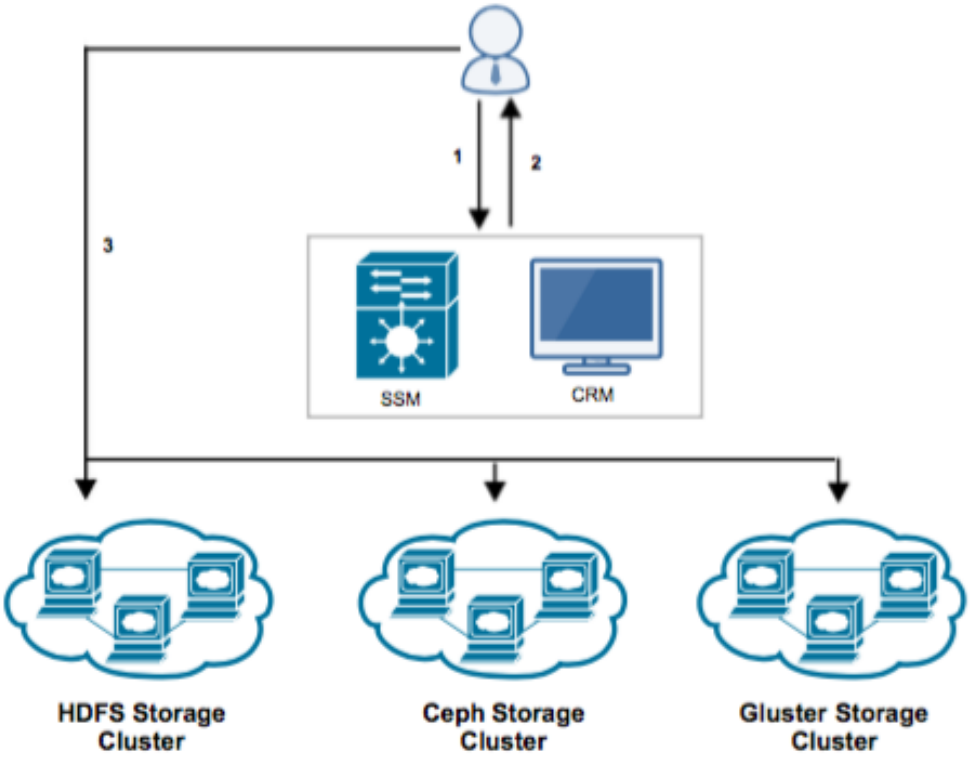

Figure 3.1: HetStore Architecture

load provisioned and new incoming workloads sent by the user. Any new SDS that is to be incorporated into the platform needs to register itself in the CRM module. CRM also incorporates a logging server.

In our architecture, *SDS clusters* consists of three different SDS backend - Ceph, HDFS and GlusterFS. We have described the configuration of our test clusters in the Evaluation section. HetStore exposes a plug-and-play feature for adding any new SDS backend. We just need to register the new storage gateway with the controller. The controller would then run a set of synthetic workloads offline to develop an analytical profile for the workload. This model would then be integrated into the existing architecture and would then start provisioning online workloads.

The *HetStore Client* consists of light weight daemons that run on each of user machines. They direct any new incoming IO workload request to the controller which replies back with the assigned SDS Cluster. The HetStore Client then directs the subsequent IO requests to the assigned cluster.

As shown in Fig 1, in **Step 1**, the users submit a API request to store a workload. The request is submitted to the Controller layer. The CRM logs all the input requests submitted by the user. The SSM stores all the data retrieved after running regression on the training set for all the storage clusters. This part is explained briefly in the SSM Algorithm section. So, when SSM receives the input request, it asks CRM for current network metric and the resource allocation of the clusters. It then decides in which cluster this test workload will be stored based on the prediction of Throughput and IOPS. It send back a response which contains the information of the cluster like IP address and type of the cluster which is **Step 2**. After receiving the response, the data is transferred through a gateway from the user side to the destination cluster as shown in **Step 3**. One thing to note, data is transferred directly from user to the cluster instead of going through the control layer which would have added to the delay.

We have created two APIs, Workload API and Storage Cluster API. Storage Cluster API is only for the administrator, as he can decide whether to add/delete a cluster. The end user can communicate with the system by submitting a workload through the Workload API. The user would have to submit a HTTP request including a json text or file consisting data about the whole workload which is to be simulated. The parameters needed would bethe absolute location of the file, size of the file and the type of IO operation. The format to simulate a workload is as follows:

```
{
  workloads: {
               id:1
               fileSize:120k
               filePath: <absolute path of the file>
               IO: read
             },
             {
               id:2
               fileSize:5120k
               filePath: <absolute path of the file>
               IO: write
             }
}
```

Similarly, we have created the API for cluster management and monitoring. The administrator could create a new cluster or update an existing cluster by adding machines to that cluster. Also, the administrator can know the status of the cluster like memory utilization, health, etc. The API makes HTTP requests to the controller which would attach/detach/update a cluster as per the request submitted by the administrator. The parameters needed for this API are name, IP address of the storage gateway and type of storage cluster.The format to create a cluster is as follows:

{

name: ceph1/hdfs1/glusterFS1

storage gateway IP: 192.168.x.x

type: Ceph/HDFS/GlusterFS

}

The Table 3.1 shows the API we have exposed to the User/Admin:

API Group	**HTTP verb**	**Description**
Storage Cluster	POST	Create a cluster
Storage Cluster	PUT	Add machines to existing storage cluster
Storage Cluster	GET	List cluster(s)
Storage Cluster	DELETE	Destroy a cluster
Workload	POST	Run a workload
Workload	GET	List workloads(s)
Workload	DELETE	Stop a workload

Table 3.1: API Description

3.4 SSM Algorithm

The Storage Selection Module (SSM) allocates incoming workloads to clusters based on a statistical model that estimates Throughput and IOPS. This decision is taken on the basis of a preference criterion, encoded in a cost function, which we assume to be totally dependent on Throughput and IOPS. The module achieves this by maintaining Support Vector Regression[27] models, which predicts the likely value of throughput or IOPS, respectively, from a priori known features, for every cluster in the ecosystem. Since, our system leaves open the scope for any cost function dependent on throughput and IOPS,

we argue that the accuracy of this system will be directly dependent on the accuracy of the regression models.

It should be noted that the list of critical measurements can be extended to performance metrics beyond Throughput and IOPS. However, for the purposes of this chapter we have focussed on these two.

3.4.1 Feature Vectors

We select six a priori known features of workloads, i.e. properties that can be directly measured when a workload is received, which have a high influence on the parameters whose estimation we seek, i.e. throughput and IOPS. For instance, the *Number of files* can be used to determine the granularity of the workload, and is therefore an important characteristic in determining the allocation. This can be observed as for workloads with small files, more granular workloads perform better on Ceph against HDFS for both throughput and IOPS.

Similarly, *Read percentage* and *Write percentage* play an important role in the throughput and IOPS that a workload will exhibit on a system.

Accounting for the impact of size on performance is done by including *Total workload size* and *Median of file sizes*. And we also include *Concurrency*, or the number of concurrent requests sent as independent blocks, in order to account for the varying levels of performance against concurrency of the underlying algorithms.

3.4.2 Allocation Method

Let $C = \{c_1, c_2, \ldots\}$ be the list of clusters in the ecosystem, and $M = \{m_1, m_2\} = \{$throughput, IOPS$\}$ be the set of critical measurements, while W denotes the feature space of workloads. Corresponding to every measurement m_j, we can interpret a function $m_j : W \times C \to \mathbb{R}^+$ as denoting, for a feature vector $w \in W$ and cluster c_i, the average value of measurement when a workload with feature vector w is run on the cluster c_i.

Definition 3.4.1 *A* cost function *is defined as a function* $\}: W \times C \to \mathbb{R}^+$ *that encodes the preference criterion of the allocation, as the expected cost that would be beared if a workload was executed on a particular cluster.*

For the cost function $\}$*, the objective of allocating workload* w *can be defined as finding cluster* c_i *such that* $\}(w, c_i)$ *is minimized, i.e. determining* $\underset{c_i \in C}{\operatorname{argmin}}\, \}(w, c_i)$

And therefore the assumption that the cost function is entirely dependent on critical measurements translates to saying that for some non-polynomial algebraic functions f, $\}(w; c_i) = f(m_1(w; c_i); m_2(w; c_i); :::)$. For most polynomial, or even some other algebraic f, the accuracy of computing or estimating $\}(w; c_i)$ is trivially dependent on the accuracy of estimating $m_j(w; c_i)$, for all j.

Therefore, for each cluster c_i and critical measurement m_j, we establish a support vector machine (SVM) s_{ij}. The support vector regression model[27] s_{ij} represents a function from feature space W to the estimation of the expected value of measurement m_j, when corresponding workload is executed on c_i. Following is the pseudocode for the allocation method.

Algorithm 2 Storage Selection Module

```
1:  procedure ALLOCATE (Workload w)
2:      for c_i in Clusters_List do
3:          for m_j in Measurements_List do
4:              estimation    s_ij(w)
5:          end for
6:          test_cost(i) = f(estimation)
7:      end for
8:      if index of min(test_cost) is x then
9:          send Workload w to c_x
10:     end if
11: end procedure
```

3.5 Evaluation

3.5.1 Test Cluster configuration

We have configured 3 different clusters for our experiments, each with the configuration showed in Table 3.2.

HDFS stands for Hadoop Distributed Filesystem. HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode (master) that manages the file system namespace and access control. An HDFS cluster has a number of DataNodes which manage storage attached to the nodes. HDFS relaxes some POSIX requirements to enable access to filesystem. HDFS test cluster configuration is as follows:

Node name	vcpus	RAM	Hard Disk
namenode	8	8GB	20GB
datanode01	4	4GB	40GB
datanode02	4	4GB	40GB
datanode03	4	4GB	40GB

Table 3.2: HDFS Test Cluster configuration

Ceph is a distributed file system with an underlying object storage. It uses a pseudo random algorithm CRUSH[40] for data distribution over a cluster of object storage devices (OSDs). A ceph cluster has three major elements namely a POSIX compliant client, a set of OSD daemons which use CRUSH to do distributed data placement and Ceph Monitors which maintain a copy of the CRUSH map. It aims to provide a unified object, block and file storage. Ceph test cluster configuration is as follows:

Node name	vcpus	RAM	Hard Disk
mon01	8	8GB	10GB
mon02	4	8GB	10GB
mon03	4	8GB	10GB
osd01	4	4GB	40GB
osd02	4	4GB	40GB
osd03	4	4GB	40GB

Table 3.3: Ceph Test Cluster configuration

GlusterFS is Network File System with the aim to build a highly scalable and fault tolerant storage backend. It has two major components namely the storage server that runs a glusterfsd and the clients which use with glusterfs client or mount command to mount the exported filesystem. GlusterFS test cluster configuration is as follows:

Node name	vcpus	RAM	Bricks (Volume Attached)
node01	4	8GB	40GB
node02	4	4GB	40GB
node03	4	4GB	40GB

Table 3.4: GlusterFS Test Cluster configuration

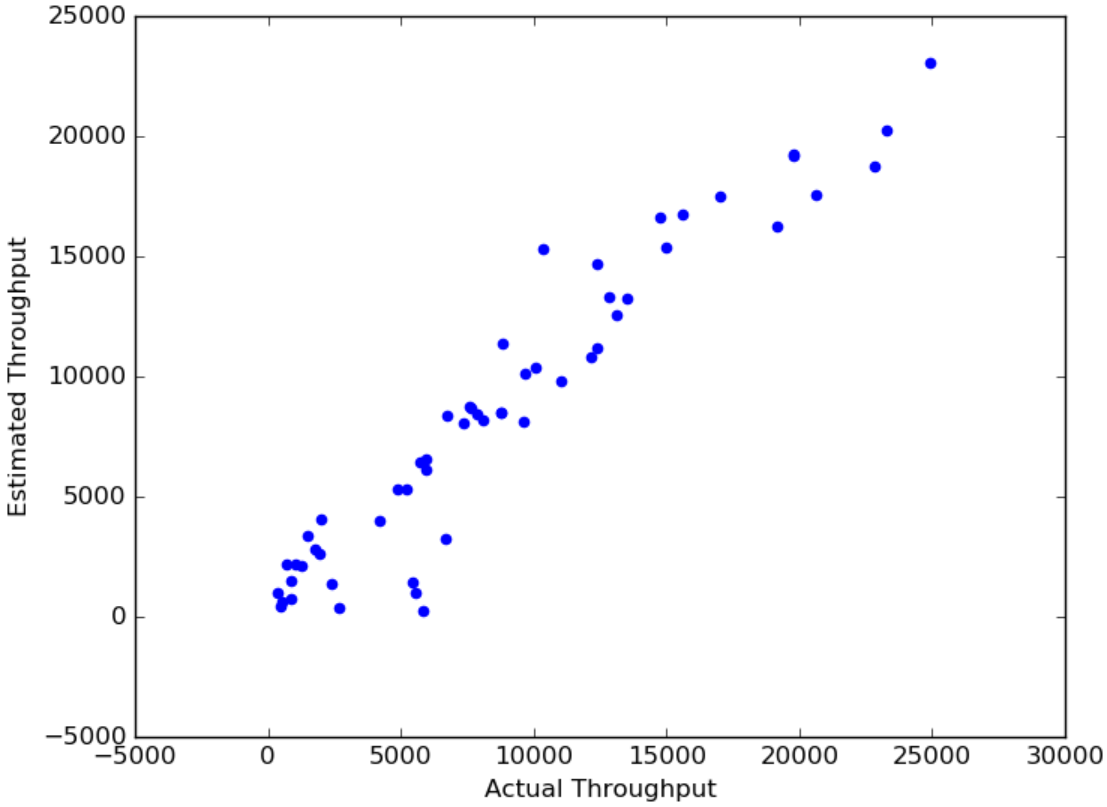

Figure 3.2: Hadoop Throughput

We also have a controller node where our two modules: SSM and CRM, runs. We need higher configuration for the controller node as there are several processes running in the same machine, also we have a Secondary controller node with the same configuration running as a backup controller.

The user interacts with the controller node with APIs described in Table 3.1 and the controller node instructs the user to transfer the data to the selected cluster from the Storage Selection Module.

3.5.2 Sample Generation

FIO - Flexible I/O Tester

We have used FIO, which is a versatile IO workload generator. Fio is very flexible as it lets you run the workload you want and could also retrieve all desired units of output. Any sort of

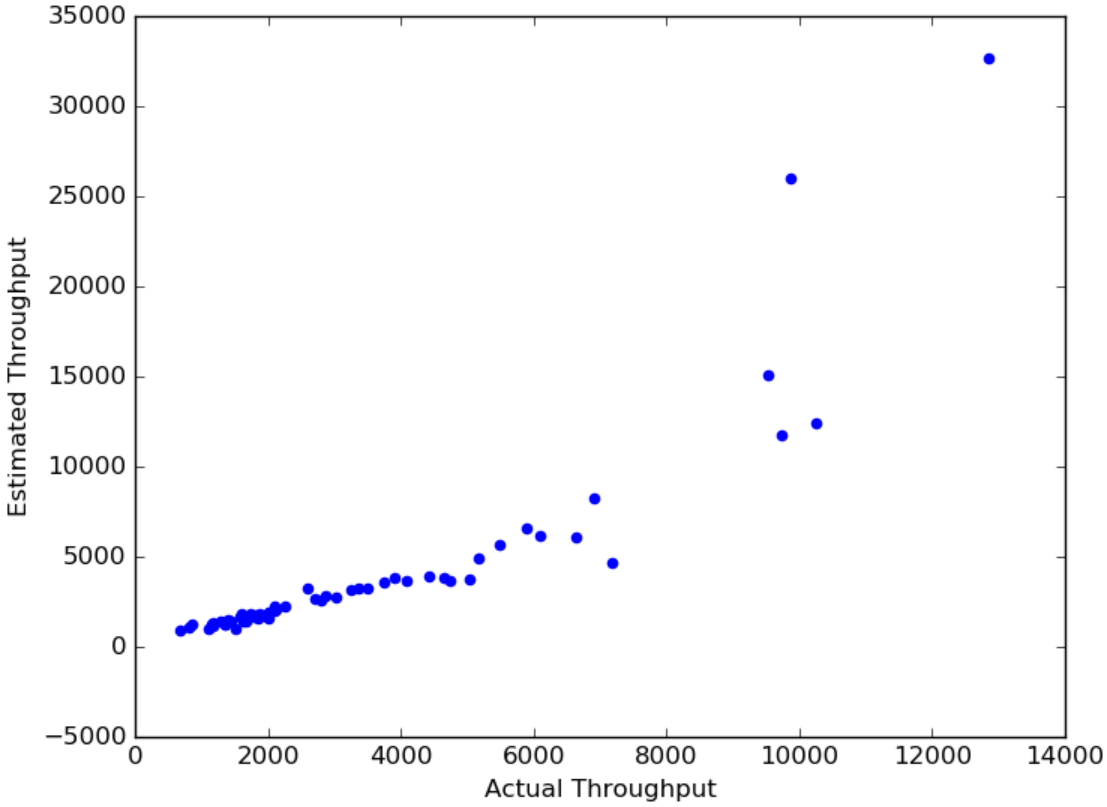

Figure 3.3: Ceph Throughput

sequential and random IO mix i.e. a read/write mix, is easy to define. Fio provides plugins to run these tests on different storage backend/ioengines. For our experiments, we have used *libhdfs*, *librbd* and *libaio* for HDFS, Ceph and GlusterFS respectively.

- *libhdfs*: It is a JNI based C api for Hadoop's DFS. It provides a simple subset of C apis to manipulate DFS files and the filesystem. This api is integrated in fio, for which we need to set the namenode and port of the HDFS cluster as parameters.
- *librbd*: It provides interface to Ceph RADOS block devices. It communicates with the librados/librbd C bindings and the krbd kernel module directly. We need to set the pool and rbd image as parameters in fio ceph configuration.

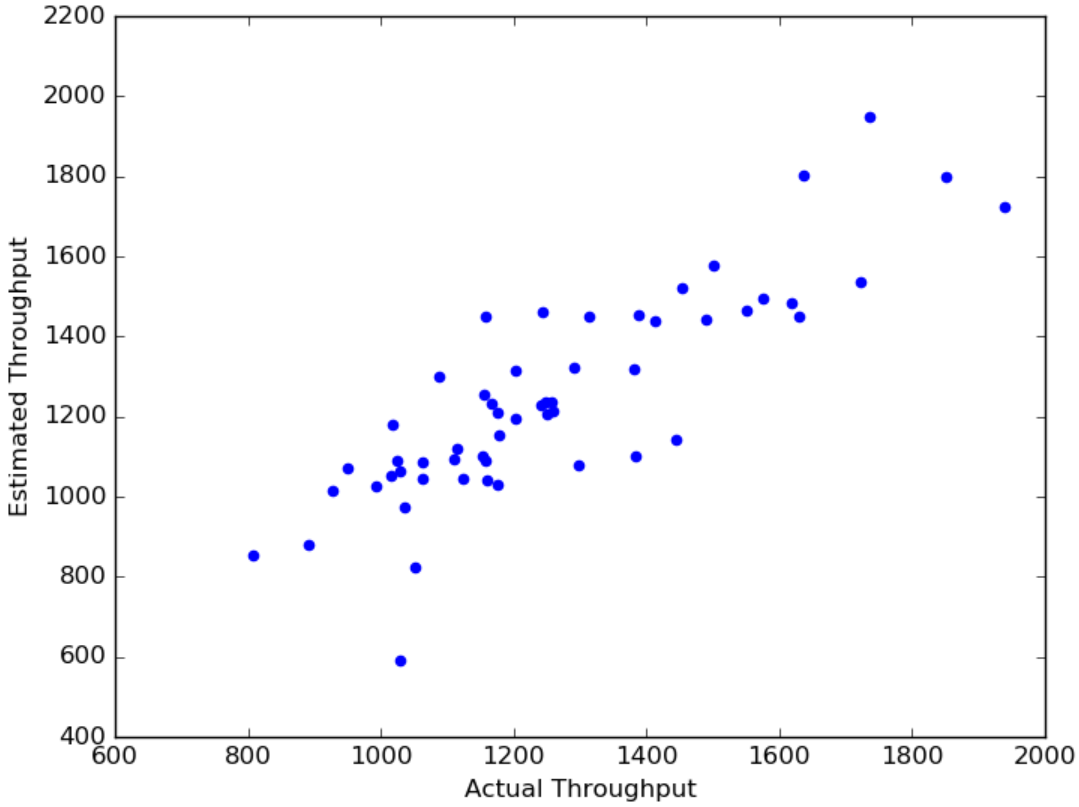

Figure 3.4: GlusterFS Throughput

– *libaio*: Asynchronous I/O (AIO) is a method for performing I/O operations on a network filesystem so that the process that issued an I/O request is not blocked till the data is available. We need to set the storage volume as parameters for libaio configuration in fio.

Workload generation

– Training data: There is a scarcity of publicly available storage dataset. To overcome we generate a diverse set of training and testing datasets. We try to include different classes of workloads like logs, transactional, photostream etc. in the mix as well as hybrid workloads.

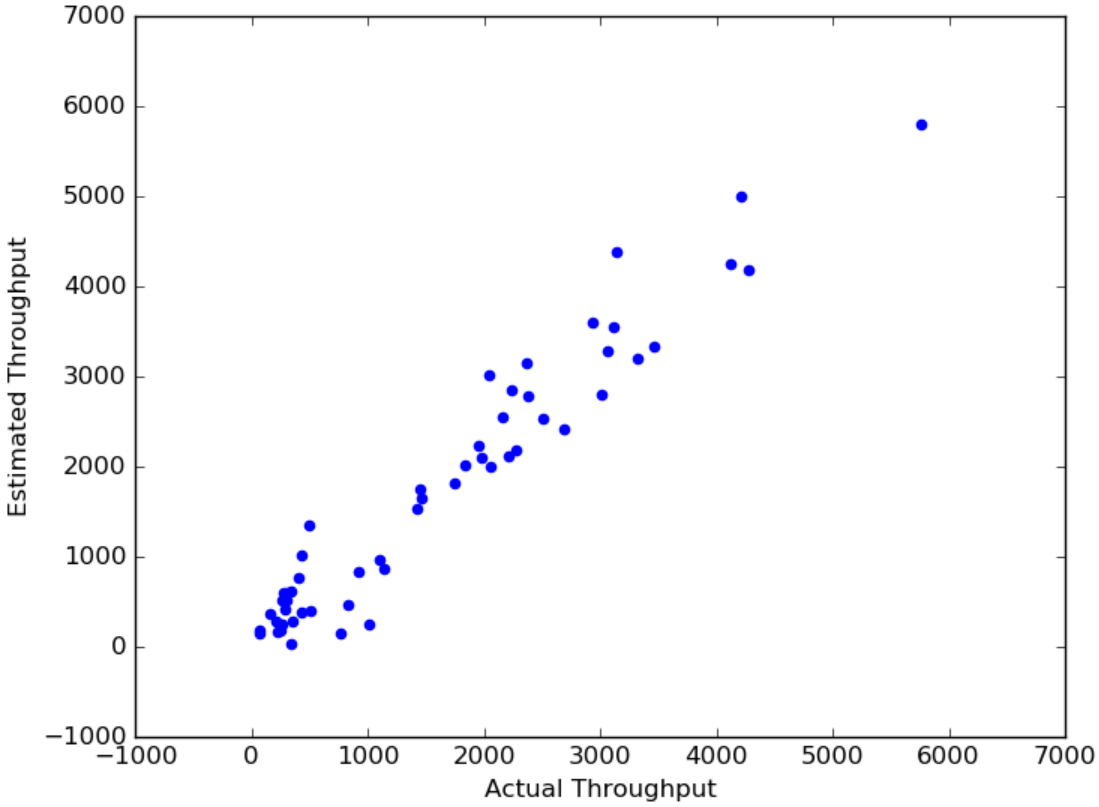

Figure 3.5: Hadoop IOPS

– Testing data: For testing purpose we generate a hybrid of IO workload classes. These workload classes are characterised by the difference of file size, read/write frequency, degree of concurrency, etc.

3.5.3 Performance of Regression

The coefficient of determination, or R^2, denotes the correlation coefficient between ground truth values and predicted values. It provides a measure of how well future samples are likely to be predicted by the model. Best possible score is 1.0, while constant model that always predicts the expected value of y would get a R^2 score of 0.0.

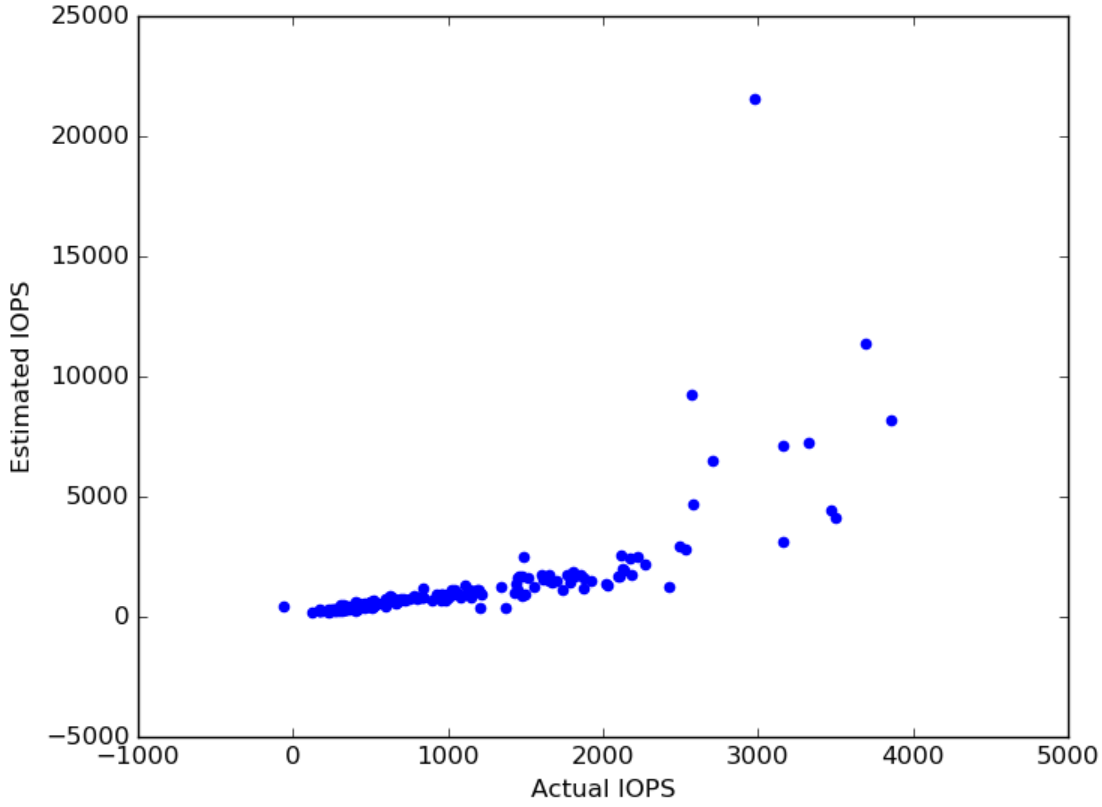

Figure 3.6: Ceph IOPS

The methodology for measuring the performance of regression is to perform 5-fold cross validation on the generated data set. That is, the data det is partitioned uniformly into 5 subsets, and the model is trained on 4 subsets before being tested on the remaining one. The average of these experiments provides us with the average R^2 score for our system.

We draw the scatter plot between the actual value and the estimated value of throughput and IOPS for our model as shown in above 6 Graphs. We also show the numerical predicted value for the same in Table 3.6. More the R^2 score, more correlated the estimated value is to the actual value.

As seen, HDFS is the most correlated followed by GlusterFS and Ceph. In each case, R^2 score is greater than 0.65 which implies it is possible to predict the value of throughput and iops with higher

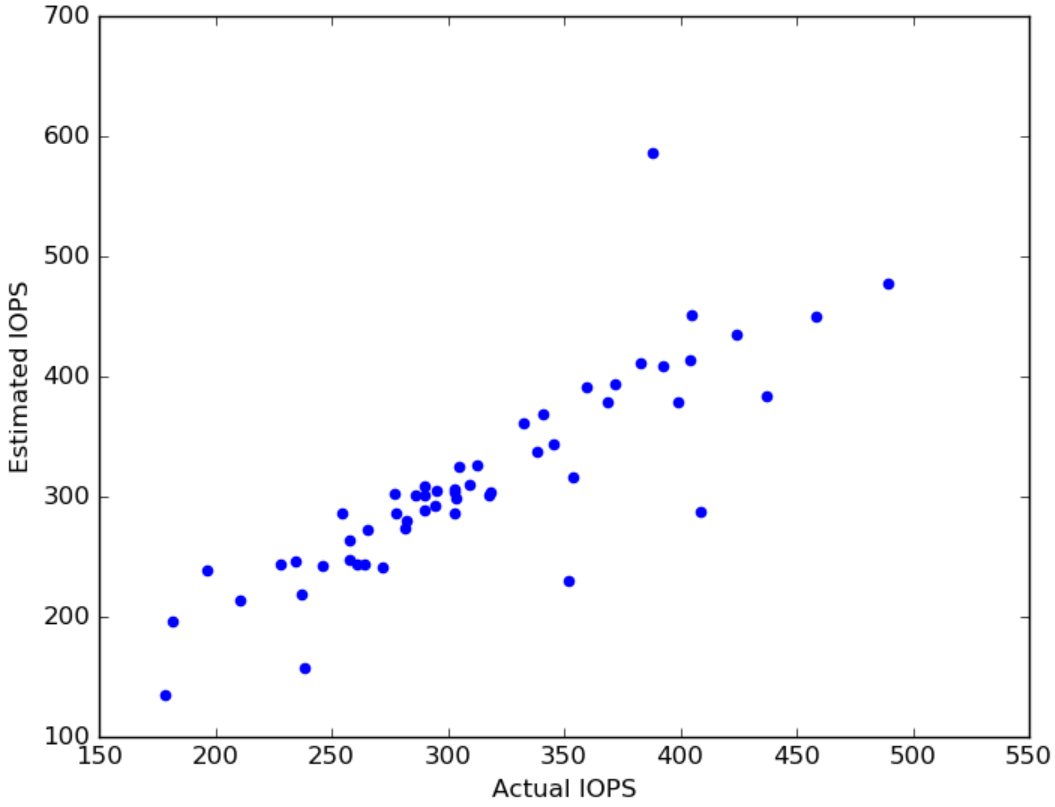

Figure 3.7: GlusterFS IOPS

accuracy, hence validating the use of statistical regression in our model since the accuracy of Assignment depends on the accuracy of predicted value by the Regression model.

3.5.4 Comparative Efficiency

Precision is defined as the proportion of instances classified in the category that truly belong to that category against the total number classified instances in that category. While *Recall* is defined as the proportion of instances classified in the category that truly belong to that category against the total number instances that truly belong to that category.

We now show the efficiency of the assignment process on the preference criteria of maximizing throughput, i.e.

Node name	vcpus	RAM	Hard Disk
Controller	8	16GB	256GB
Secondary Controller	8	16GB	256GB

Table 3.5: Controller configuration

Storage Backend	Prediction metric	Avg R^2 score
HDFS	Throughput	0.90571196245
Ceph	Throughput	0.652513723598
GlusterFS	Throughput	0.711219633794
HDFS	IOPS	0.925336270011
Ceph	IOPS	0.656942793762
GlusterFS	IOPS	0.735454555611

Table 3.6: R^2 score for different Storage backend

Cost Function = - Throughput.

For each of three classes we find the precision and recall of our platform for the testing dataset that we generated.

Table 3.7 shows the value of Precision and Recall for the Classification problem. We see the classifier predicts the HDFS class with very high precision of more than 0.9 followed by GlusterFS (with precision around 0.65) and Ceph.

However, it must be established that these values of precision and recall have been calculated for a particular uniformly generated set of samples, and are bound to be different when executed on samples from some other distribution. The performance largely depends on the distribution for sampling, and therefore the evaluation of the classification could differ in different use-cases that might result in different distributions.

As we observe, while the classifier is able to provide a decent maximization of the throughput, some limitations of the performance can be attributed to the dynamic nature of the underlying SDSs and the offline nature of our system.

Storage Backend	Precision	Recall
HDFS	0.9014084507	0.9275362319
Ceph	0.6	0.5294117647
GlusterFS	0.6363636364	0.6363636364

Table 3.7: Comparative Efficiency

3.6 Miscellaneous System Features

Fault Tolerance

A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. Here, in HetStore, we try to achieve this by spawning a backup controller. This architecture is inspired by Namenode, Secondary Namenode architecture of Hadoop. In a similar fashion, the backup controller keeps the metadata and the logs in sync with primary controller. Hence when the primary node fails, the backup controller will update the metadata and would start acting as the primary controller.

Plug and Play Mechanism to attach new SDS backend

In this chapter, we integrate three different SDS clusters namely HDFS, Ceph and Gluster. Though, the administrator is free to integrate any new the storage backend dynamically. The controller would then run a set of synthetic workloads offline to develop an analytical profile for that particular storage backend. This model would then be integrated into the existing architecture and would then start provisioning online workloads. With such a plug-and-play feature, we give full freedom to the administrator to integrate any storage backend of her choice.

3.7 Summary

The problem of providing optimal assignment for backend storage was a central problem in the design of cloud systems. It has taken a further central role as a result of growing heterogeneity from emerging Software Defined Storage systems. So, in this chapter, we proposed a solution for optimal IO Workload assignment using statistical modelling to estimate measures of performance such as Throughput, IOPS, et al. The proposed system used support vector regression to estimate the performance of individual IO Workloads on each available Software Defined Storage(SDS) system for optimal assignment.

Chapter 4

Workload Assignment in a Heterogeneous Multi-Cloud Environment

4.1 Introduction

We have witnessed the evolution in Virtualization and cloud computing since the last decade. A lot of things have changed, as traditionally physical servers and storage were used and people had to manually configure it to run their job. This process was tedious and had a lot of overhead as researchers or small scale businesses had to manually setup their infrastructure to run their workloads or application. Also, the cost incurred was quite high due to network cost, storage cost, electricity utilization and manual effort. But times have changed, companies like IBM, Amazon, Google, Microsoft, etc started providing Infrastructure-as-a-service which included various services like virtualization, storage solutions, Virtual Private Network (VPN), network and many many more exhibiting characteristics like agility, stability, reduced cost, performance, security etc. So, as a customer, you would have to pay for the services you need and relieve oneself from the extra overhead which traditional computing offered.

Before companies like IBM offering the whole computing stack would charge you for the same, but after the advent of virtualization and cloud computing, a new model gained attention called pay-as-you-go model (PAYG). Here the customer had to pay only for the resources they use, as they use them. This is a great model as you don't have to worry about any commitments or contracts. As different public provider started adopting this, they also started to make different pricing model to differentiate itself from other existing solutions. In this chapter, we would leverage the unique features of a public cloud provider in a multi-cloud heterogenous environment for IO workload assignment. We use Google Cloud Platform(GCP), Amazon Web Services(AWS) and Microsoft Azure as our cloud providers to evaluate our technique.

In a multi-cloud setup, the heterogeneous architecture use multiple cloud services to increase flexibility, cost optimizations, increase efficiency (throughput, iops, latency), etc. As stated, we would use

Google Cloud Platform, Microsoft Azure and Amazon Web Service, and we know that each public provider has its own characteristics like different availability zone, services, pricing model, unique discount schemes, frequent price cut, etc. Also, fluctuations are seen in price and efficiency with same type of services and same resources but with different cloud provider. Here are some unique features of the following clouds which we would be using in our heterogeneous architecture: In respect to pricing, Google Cloud Platform applies automatic discounts using Sustained Usage Discounts(SUDs), preemptible VMs and ability to make a custom instance, Amazon Web services provides schemes such as Reserved Instances (RIs), Spot Instances, Spot Block and Fleet and off-peak price cuts whereas Microsoft Azure schemes depends on your Enterprise Agreement(EA) which is a volume licensing package. Also, with respect to efficiency, the underlying hardware/resources used and the product stack for the services are quite distinct. We plan to leverages all these schemes and qualities of these clouds and inculcate them for a better assignment and minimizing cost. But comparing apples-to-apples is difficult in this scenario as all these have different characteristics which also depends on real time user data. So, we have designed and developed "CM(EM)", which would increase your workload performance along with optimization in cost reductions, taking into account the real time data also.

In our multi-cloud heterogeneous framework CM(EM), we first try to find the right class for instance and storage for which a particular workload would perform well for each cloud. This is done by the Class Finder Module, where a training model is created and user submit their workload query with some parameters and the classifier gives us a class where the user submitted workload would perform the best. Then we filter some parameter from the user submitted query and some extracted parameters which would help us design cost minimization techniques for the same. This technique would apply discounts and schemes whenever applicable for a given cloud and returns us the instance with required configuration for the workload with the minimum cost amongst the evaluated clouds.

Here, we are trying to solve two problems: 1) First one is the optimal assignment of IO workload to their respective instance/machine type and storage type for a given cloud to gain maximum efficiency ii) Cost minimizations which gets the instance/storage type as input and tells us which instance/storage to pick to gain maximum discount. So for evaluation of our system:

Empirical Accuracy of prediction model(Validation): We generate a synthetic training set and evaluate our estimator/classifier by cross validating the evaluated training sample through R^2 score which denotes the coefficient of Determination. Also, we cannot evaluate the clustering due to lack of public dataset for IO workload assignment.

Cost comparison for the multi-cloud environment: We compare the total cost incurred for a workload to run for our system with respect to a baseline cost for resultant instance class and storage class (from the classifier) for each cloud provider.

4.2 Related Work

Ang Li and Xiaowei Yang et al.[32] provides metrics for making public cloud selection phase easy. They use some widely used non-cloud contexts to benchmark i) task's runtime, to measure a virtual instance's efficiency, ii) a storage service's latency and throughput, and iii) a network path's latency and capacity. In this chapter, we assume the intra network cost to be negligible as data transferred between instances of same zone is generally free, and as we are measuring performance of workload IOs, it mainly depends on the system and the storages performance.

Ruben Van den Bossche et al.[37] demonstrates the power of using hybrid cloud setting to assign tasks in an application. His objective is to maximize the utilization of the internal data center and to minimize the cost of running the outsourced tasks in the cloud, while fulfilling the applications' QoS constraints. They characterize their workload by CPU, memory and data transmission requirements on a preemptible and deadline constrained hybrid cloud environment. They propose a Binary Integer program formulation to this scheduling problem and evaluates computational cost with respect to the parameter involved.

We have done previous work in Software defined storage(SDS), where we propose a framework called HetStore for optimal IO workload assignment using statistical modelling to estimate the performance such as throughput and latency. This modeling was done using a heterogeneous storage environment using Ceph, HDFS and Gluster. The accuracy of estimation of throughput and latency achieved by the above modelling with value of coefficient of determination over 0.65 in all cases.

4.3 Architecture and Implementation

From Figure 4.1, we see that our Architecture is divided into 3 Modules: Class Finder Module, Filtering Module and Cost Calculator. We will explain each of these modules in this section.

4.3.1 Class Finder Module

We have Measurement Prediction, where we predict the Virtual machine performance parameters and Storage performance parameters for a particular cloud provider. In Workload Time Estimator, we predict the time taken by a workload to run in a particular cloud provider. We do statistical modelling to predict those values, and then we label/assign the training data with the Instance class and the Storage class by assigning a score through a decision function. We then do Clustering on the training data, and then we assign the class to the testing data through the output of clustering.

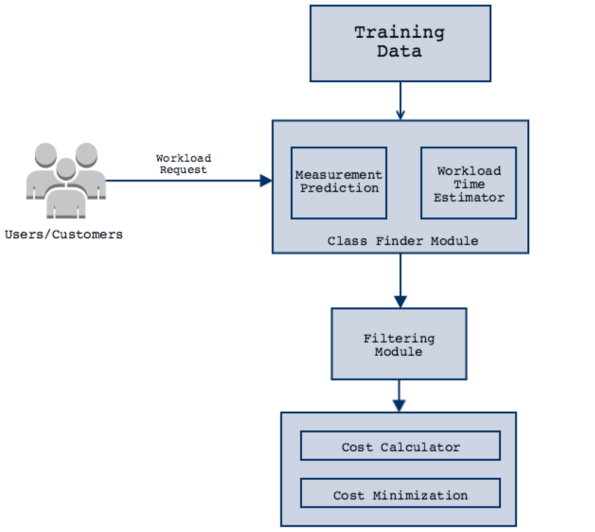

Figure 4.1: CM(EM): Pipeline Architecture

Definition 4.3.1 *For C as the set of generic classes received from clustering of the performance parameters of the training data, i.e. feature vectors in a space W. And for every task $t \in T$, we use statistical prediction from $\}: T \to W$, we estimate the corresponding performance vector $w \in W$, and then assign it a corresponding class in C.*

4.3.1.1 Statistical Regression and Modelling

We have different parameters for measuring a workload's performance on a virtual machine and storage. The module predicts the performance measurement by maintaining Support Vector Regression[22] models, which predicts the likely value of those measurements, respectively, from a priori known features, for every virtual machine class and storage class in the ecosystem.

Feature Vectors

We choose 6 workload features/characteristics, 3 Instance/VM features and 1 storage feature, which can be directly measured when a workload is received and which has a high influence on the parameters whose estimation we seek. So we have a total of 10 features, which we would use in our space for statistical regression. The 6 workload features are: *Number of Files* in a workload used to determine

workload granularity, *read percentage* and *write percentage*, *total size of the workload* and *median of file sizes* accounting for the impact of size on performance and *concurrency* i.e. the number of concurrent request sent as independent blocks. The 3 instance features are: *vCPUs*, *RAM/internal memory* and *processor*, and the storage feature is *Memory* of the storage used.

Annotating the training data

The parameters used in our architecture to measure the virtual machineś performance are *CPU performance*, *network bandwidth* and *local memory bandwidth*. Similarly, for measuring the storageś performance, we use *Throughput*, *IOPS* and *latency*. We do not have any direct relation or any straightforward equation to measure the VMs or storages performance, but we know the impact (i.e. positive/negative) that a parameter would have on its performance. For a given workload from the training set, we run the workload on all virtual machines of all instance type (Table 4.1) and all the corresponding storage from the storage class (Table 4.2), and extract all the performance parameter using various benchmarking tools available. So, we take the normalized parameter and give equal weights to them to calculate a score which would help us in determining the given workload should be assigned to which Instance and Storage class. The workload is assigned to a particular storage and vector class, where this above score is the highest. So, for our training data, we use this scoring function to annotate it i.e. assign an instance and storage class for that training workload. For Instance type, the scoring function is:

$$S_{vm} = \hat{t}T + \hat{\imath}I \quad \hat{l}L$$

where, T = Throughput(MB/s) of the workload, I = IOPS(operations/s) of the workload and L = latency (ms), and $\hat{t}$, $\hat{\imath}$ and $\hat{l}$ and their normalized vectors respectively.

Scoring function for storage class is:

$$S_s = \hat{c}C + \hat{n}N + \hat{m}M$$

where, C = CPU performance(programs/execution-time), N = Network Bandwidth(Bits/sec) and M = Local Memory Bandwidth (Bytes/sec) of a workload for a storage class and $\hat{c}$, $\hat{n}$ and $\hat{m}$ are their normalized vectors respectively.

Applying SVM for predicting measurements and WTE

After annotating the training data, we use support vector machines to train the vectors to its corresponding performance measurements and then use the regression model to predict these measurements for a user submitted workload i.e. the testing data. Workload Time Estimator (WTE), also works using the same principle, but here the measurement is time, and we predict the execution time of a given workload on all the public cloud providers.

4.3.1.2 *k-Means* Clustering

We use the standard k-means clustering algorithm (iterative refinement technique), where n features points having d-dimensions, we cluster these points into k clusters. For assigning these points to the clusters, we use the standard distance function for a d-dimensional space as our objective function. k-means comprises of mainly two steps: i) Assignment ii) Update. In the Assignment step, each vector is assigned to the cluster, and then the centroids are updated in the second step, and the process is repeated until convergence when the assignments no longer change.

Instance classes/types

We will explain each instance type for each cloud in more detail in this section. Table 4.1 contains list of class and the type of instances for each cloud in that class:

VM class	**AWS instance type**	**Azure instance type**	**GCP instance type**	**Rationale**
General Purpose	t2, m4	A0-7, Av2, D1-4 v1, D1-5 v2	n1-standard-*, f1, g1	t2, m4, A0-7, Av2, D1-4 v1, D1-5, f1, g1, n1-standard-* are low cost general purpose instance type
Compute optimized	c4	F, H	n1-highcpu-*	c4, F, H and n1-highcpu-* are compute and memory optimized
Memory optimized	x1, r4, d2	D11-14 v1, D11-15 v2	n1-highmem-*	x1, r4, d2, D11-14 v1, D11-15 v2 and n1-highmem-*optimized are for storage performance and high IO tasks
GPU optimized	p2, g2	NC, NV	External GPU	p2, g2, NC, NV and External GPU are optimized for gpu compute applications

Table 4.1: Instance Class and corresponding images in all clouds

Standard/General Purpose

AWS's General Purpose Instance type provides baseline CPU performance with the ability to burst and also provides a low-cost platform for consistent processing performance. Microsoft Azure's general purpose instance type provides a powerful combination of local disk, CPU and memory used for production applications, web applications, low-traffic websites, build servers, etc. Google Cloud platform's standard instance type provides bursting capabilities allowing physical CPU to be attached for short period of time. A cost efficient type for running small and non-resource intensive application are the main use case for this machine type.

Compute Optimized

AWS's compute optimized type provides very high compute performance compared to any other instance type in EC2. They are used for compute intensive workloads like network appliances, batch processes etc. Microsoft Azure's compute optimized instances has local SSD attached of 16 GB per CPU core mainly used in high performance clusters, simulation, modeling and other network intensive scenarios. Google Cloud Platform's compute optimized instance type provides one virtual core of every 0.9 GB of RAM and are used for tasks requiring more CPU relative to memory like scientific or financial modeling, video encoding etc.

Memory optimized

AWS's memory and storage optimized instance type provides very high disk throughput for large datasets requiring high sequential I/O performance. They are used for distributed web scale in-memory caching, database and real-time processing, etc Microsoft Azure's memory optimized instance type provides very high memory to core ratio ideal for running medium to large scale cache, RDBMS servers, in-memory processing, etc Google Cloud platform's instance type has 6.5 GB of RAM per virtual core and are great for running caching servers, in-memory analytics and many other tasks requiring more memory than virtual CPUs.

GPU optimized

AWS's GPU optimized instance type provides high network speed and high CPU performance along with general purpose and graphics GPUs for computational finance, molecular modeling, 3D graphic rendering, etc. Microsoft Azure's GPU optimized instance type provides specialized VMs for high computing and performance ideal for heavy graphic rendering, video editing, etc Google Cloud Platform does not provide a separate instance type for GPU optimized workloads, but they provide separate GPUs which can be attached to any instance type. We add an additional cost for the GPU with the attached VM for the classification and price modeling.

Storage classes/types

We will explain each storage type for each cloud in more detail in this section. Table 4.2 contains the list of class and the type of storages for each cloud in that class:

SSD Storage

AWS's provides high performance SSD volume designed for a wide variety of transactional workload used for IO intensive NoSQL and relational database, low-latency applications, etc. Microsoft Azure provides Premium Managed SSD disks for high performance to support I/O intensive workloads with low latency and high throughput. You can provide the size and the number of disks required which provide different throughput caps, input/output operations per sec

Storage class	AWS	Azure	GCP	Rationale
SSD Storage	gp2, io1	Premium managed storage	Local SSD	gp2, io1, Premium managed storage and Local SSD are general purpose SSD volume that balances price and performance for a wide variety of transactional workloads
HDD storage	st1, sc1	Standard managed disks	Persistent Disks	st1, sc1, standard managed disks and persistent disks are low cost HDD volume designed for frequently accessed, throughput-intensive workloads
Object Storage	S3	Blob Store	Cloud Storage buckets	S3, Blob Store and Cloud Storage buckets are object storage system with a simple web service interface to store and retrieve any amount of data from anywhere on the web

Table 4.2: Storage Class and its types in all clouds

(IOPs) and monthly price per GB. Google Cloud Platform provides Local SSDs which have high throughput and low latency, are attached physically to the servers that hosts the VM instance.

HDD Storage

AWS provides low cost HDD volumes which are designed for throughput intensive workloads and frequently accessed tasks used in log processing, big data mapreduce jobs, data warehousing, etc. Microsoft provides Standard Managed Disks using HDD which are ideal for development/testing and other infrequent access workloads which are less affected by performance discrepancy. Google Cloud Platform provides persistent disks which are similar to physical disks in a server with respect to its functionality which are quite durable. GCP manages the whole stack to optimize performance and ensure data redundancy.

Object Storage

AWS provides S3 - Simple Storage Service which is an object storage, which makes web related computing quite easy and affordable for developers. It provides highly scalable, fast, reliable and inexpensive data storage infrastructure which Amazon use itself also. Microsoft Azure uses Block storage as it object storage environment for a cost efficient and scalable solution. It is ideal for storing document and social data content like images or files for web applications, backup files, logs, large datasets, etc. Google Cloud Platform uses Google Cloud Storage buckets which are

quite flexible, scalable and durable options for object storage. It is ideal for situation where the application does not need low latency.

4.3.2 Filtering Module

The filtering module uses the user submitted parameters as well as some extracted parameter from the system as well. We will explain both these cases and explain how could we condition our results. We will explain some implementation details as how a user can submit a job, and how can an admin keep track of all the workload request. CM(EM) provides with two APIs, User API and Admin API. The parameters required from the users are: id, job/task priority of the job and the workload itself. A user submits a workload request by giving the parameter in form of a json request explained in Table 4.3.

API Group	**HTTP verb**	**Description**
User API	GET	Returns a list of all workload request by this user
User API	CREATE	Create a workload request
User API	DELETE	Delete a workload request
Admin API	GET	Returns a list of all workloads submitted by all users
Admin API	DELETE	Admin can terminate any workload request submitted by a user

Table 4.3: API Description

4.3.2.1 Extracted parameters

We also try to model based on some of the extracted parameters from the workload request submitted by these user. The *workload feature vector* is extracted from the job submitted by the user, which is then used in statistical modelling i.e Class Finder module, for predicting the class the workload performs best in. *Location* from which the user has submitted his job and also the *date* and *time* of the job. These are important parameter which will be used in the Cost Calculator for calculating the price of the chosen instance. Also, we extract the *estimated_execution_time* for the workload to run for each cloud provider given the workload feature vector by predicting the approximate time for the workload to run in Workload Time Estimator (WTE), which is updated in real time i.e we keep adding the actual

workload time of the user submitted workload when it is completed to our framework to increase the accuracy of prediction by adding real point to our model.

Some rules and conditions extracted from this module, which will be used in our Cost Minimization phase:

1. If *estimated_execution_time* > 6 hours or *priority* == "high", then we cannot include spot instances of AWS for cost minimization, as they are sensitive to failures i.e. termination.

2. If *estimated_execution_time* > 24 hours, then we cannot include preemptible instances of Google Compute Engine, as they are also sensitive to termination.

3. We use the *location*, date and time of the workload to calculate off-peak hour discounts for the zone/location and time.

4.3.3 Cost Calculator

This module will collect all the pricing data i.e price history of instance, current cloud prices, off peaks prices and calculates the price of the workload, which is the total cost of the compute resource, storage and network. In this part, we will explain the different discount schemes provided by each cloud in detail, and generate cost functions to calculate the minimum cost.

4.3.3.1 Discount schemes for compute

We explain all the discount schemes for all the clouds in this section.

Amazon AWS

Spot Instances
Amazon EC2 has a Spot market where the customers can bid for spot instances, which are of a very low cost compared to On-Demand instances. These instances are available for a period of 1 to 6 hour max, and can be terminated in between. Amazon allows you to bid on spare EC2 computing capacity through its bidding portal and also provides an API to do it. The price of EC2 instances changes real time as its spot instances are based on supply-demand model. It provides discount upto 70%-80% compare to On-demand price.

Spot Block and Spot Fleet
Spot Block and Spot fleet are also spot instances but recently Amazon has this new concept where

if a user want to run his workload for a specific amount of time which is less than 6 hours, you can block your VM from being terminated and prevents data loss. Also, you can also specify the number of instances you want to reserve or block through Spot Fleet, where you place your bid and the number of instances you require and spot fleet will schedule it for you. The price of spot block instances are 30% to 45% less than On-Demand. It also provides the users with an additional 5% off during non-peak hours for the region.

Reserved Instances(RIs)
AWS offers Standard Reserved Instances for 1-year or 3-year terms. So these instances are useful for long term processes but offers significant discounts (40-60%) and great schemes. We do not consider this parameter in our cost calculation and optimization as we cannot predict or create user history in this scenario.

Microsoft Azure

Under the Microsoft Enterprise Agreement (MEA), organizations with 250 or more users are extended best deals in terms of flexibility to buy cloud services and software licenses under one agreement. They provide 15-45% discount depending on various factors like size of the org, resources required etc

Google Cloud

Sustained use discounts
Google provides discount on basis of usage and are applied automatically in the monthly billing. When an instance spawned by a user is used for more than 25% of the time, Google Compute Engine provides discount, for every incremental minute of the usage. For instances used 25%-50% in the monthly billing, then 20% discount is applied, if 50%-75% time used, then 40% discount is applied and more than 75% of the time, then 60% discount is applied. GCE also performs cost optimizations itself by combining multiple and non-overlapping instances which are in the same zone to be considered as a single instance to gain maximum discount.

Preemptible VMs
Google provides preemptible VMs, similar to spot instance, but there is no bidding involved in Google's pricing model and it is not market sensitive. Preemptible VMs are short lived, highly affordable compute instance for fault tolerant workloads and batch jobs/tasks. These instances last upto 24 hours and they are not available always as there are finite compute resources. If your

application is fault tolerant and can withstand termination, then this is a great choice as a straight forward discount of 80% is applied. The probability that the Compute Engine will terminate your instance before 24 hours is generally low, but might vary depending on the day and the zone.

Custom instances
Workloads that are not a good fit for the predefined machine types that are available to you(upto 40% decrease in cost than general machine type). Workloads that require more processing power or more memory, but don't need all of the upgrades that are provided by the next machine type level.

4.3.3.2 Discount schemes for Storage

There are no schemes or discount of any sort with respect to storage of any type in a cloud, but are instance type with local SSD attached with the VM, so extra cost of the storage is saved if the optimal storage class predicted by the classifier is local SSD storage type. We have a mapping of all the storage types of each cloud as ids and their cost as its value. The Cost Calculator uses this predefined map to calculate the cost incurred to the user for the submitted workload.

4.3.4 Cost Minimization

In this phase, we try to minimize our total cost for the chosen Instance Class and Storage class from classifier i.e. the summation of instance cost, storage cost and network cost. This can be achieved by minimizing those individual cost parameters itself.

Definition 4.3.2 *For every storage class* $g \in C_{storage}$ *and instance class* $h \in C_{instance}$*, there exists a cost function that can be parameterized as :*

$$\textit{Cost Function} = C^{g;h} = \min_{i;j}(C_{vm}^{g;h}(i;j) + C_{s}^{g;h}(i) + C_{n}^{g;h}(i))$$

where, i = *public cloud provider*

j = *all zones for a particular cloud provider*

$C_{vm}^{g;h}$, $C_{s}^{g;h}$ *and* $C_{n}^{g;h}$ *are cost function, in storage class* g *and instance class* h*, for virtual machines, storage and network respectively.*

(Hereafter in this chapter, we assume the cost functions to refer to the specific cost functions for the determined storage and instance classes, and shall ignore g and h for ease of notation.)

Compute/Instance price calculation (C_{vm}):

All the public providers have different schemes and platforms for providing compute resources to customers and users. In this part, we try to leverage the schemes mentioned in the previous part, and give the user the maximum discount for the best instance and storage class he has chosen from. We also find the optimal configuration which is required from the instance by estimating the memory and the virtual CPUs required. We need to calculate the minimum of cost incurred from all the cloud with respect to all its zones.

$$C_{vm} = \min_{x;y;z}(C_{awz}(x), C_{maz}(y), C_{gce}(z))$$

where, x, y and z are all the zones for the particular cloud provider and C_{awz}, C_{maz} and C_{gce} are the cost functions for Amazon Web Services, Microsoft Azure and Google Compute Engine respectively.

$$C_{aws} = \min_{x}(C_{si}(x), C_{sb}(x))$$

where, C_{si} and C_{sb} = cost function for computing price of spot instance and spot block/fleet respectively for a given zone x.

We will explain the cost functions for both these above parameters in pseudo code. Bahman Javadi et al.[29], applied statistical modelling on predicting the spot bid from the previous market history of spot bids. He analyzed all different types of spot instances with respect to current spot price and inter-price time (i.e. the time between price change) and determined the time dynamics for spot prices in hour-in-day and day-of-week. We use this to estimate the appropriate spot bid to be placed to increase the chance for getting the instance. Also, Spot block and Spot instance have different bidding framework, as they have different characteristics.

We explain pseudo code for C_{si} and C_{sb} in Algorithm 3 and Algorithm 4 respectively.

$$C_{maz} = C_i(y) - EA(y) * C_i(y)$$

Here, EA(y) = constant, represents the Enterprise agreement License scheme, which is a contract for a term where Microsoft provides discount(%) as mentioned in the discount schemes, and C_i = cost of chosen Azure instance cost for zone y.

$$C_{gce} = \min_{z}(C_{sud}(z), C_{pi}(z))$$

Algorithm 3 Amazon Spot Instance Cost

```
function SPOT REQUEST (Instance class ic)
    for z_i in Zone_List do
        if estimated_execution_time < 6 then
            if priority is {low,medium} then
                cost   spot_bid(ic)
            end if
        end if
        if request_spot = SUCCESS then
            C_si   cost
        else
            C_si   base_price(ic, z_i)
        end if
        return C_si
    end for
end function
```

Algorithm 4 Amazon Spot Block/Fleet Cost

```
function SPOT BLOCK REQUEST (Instance class ic)
    for z_i in Zone_List do
        if estimated_execution_time < 6 then
            if priority is {medium, high} then
                cost   spot_block_bid(ic)
            end if
        end if
        if request_spot = SUCCESS then
            if off-peak_discount then
                cost   0.95 * cost
            end if
            C_sb   cost
        else
            C_sb   base_price(ic, z_i)
        end if
    end for
    return C_sb
end function
```

For Google Compute Engine, sustained use discounts(SUDs) are applied automatically in the billing month, but from knowing the history, we could apply discount ourselves to know the exact price for the instance to run the user's workload. We take the history of the current billing month, and the instance type, and calculate the usage of the instance in that period(as discounts are also applicable on inferred instance). In Preemptible VMs, Google Compute Engine gives 80% straight discount on instances, if available. The pseudo code for calculating discounts for Google's SUD and Preemptible instances are explained in Algorithm 5 and Algorithm 6 respectively.

Algorithm 5 Google SUD Cost

```
function GOOGLE SUD (Instance class ic)
    for z_i in Zone_List do
        C_sud  base_price(ic, z_i)
        if usage(ic)  0:75 then
            return 0:4  C_sud
        end if
        if usage(ic)  0:50 then
            return 0:6  C_sud
        end if
        if usage(ic)  0:25 then
            return 0:8  C_sud
        end if
        return C_sud
    end for
end function
```

Algorithm 6 Google Preemptible Instance Cost

```
function GOOGLE PREEMPTIBLE (Instance class ic)
    for z_i in Zone_List do
        if estimated_execution_time < 24 then
            if priority is {low,medium} then
                place_request(z_i)
            end if
        end if
        if request = SUCCESS then
            C_pi  0.2 * base_price(ic, z_i)
        end if
        return C_pi
    end for
end function
```

Storage price calculation (C_s):

As mentioned earlier, we store the mapping of all the storage type of all cloud to their corresponding cost. We use the below formula to calculate the storage cost for a workload:
(cost in per GB-month * Total Storage used in GB * total time used in hours / (24 (hours/day) * 30 day-month)).

Network price calculation (C_n):

As mentioned earlier also, there is no scheme related to network cost and we see very less difference between the network rates between all public clouds.

4.4 Evaluation

4.4.1 Sample Generation

Training data: There is a scarcity of publicly available storage dataset. So we generate a diverse set of training and testing datasets. We try to include different classes of workloads like logs, transactional, photostream etc. in the mix as well as hybrid workloads. We manually label the workload vector of the training data by using some heuristic explained in the Architecture section.

Testing data: For testing purpose, we generate a hybrid of IO workload classes. These workload classes are characterised by read/write frequency, difference of file size, etc. Also, the user has to submit some parameters for the Filtering Model, we also randomize them for our training and testing phase both.

4.4.2 Tools

We have used *iperf3*[16] as our tool to measure the performance of a virtual machine in any cloud provider. The parameters extracted are: RAM performance, CPU performance and network performance. *iperf3* runs some benchmarking scripts to calculate the local memory bandwidth, network bandwidth and the cpu performance when utilized i.e during the workload.

FIO[9] is a IO workload generator and performance benchmarking tool for storages. Fio is very flexible as it will let you run your workload and you could also retrieve all desired units of output like read-write aggregate throughput(or latency) and latency. Also, throughput and latency are getting a lot of importance, especially, when it comes to high sequential data transfer workloads such as video or audio edition environments, big data workloads, data warehouse queries, in a multi-cloud environment.

4.4.3 Performance of Regression

The correlation coefficient between ground truth values and predicted values is denoted by R^2, the coefficient of determination. We calculate the score on how well the future sample are predicted by the model. The best possible score is 1.0 and a minimum score 0.0, the predicted expected value when there is a constant model. We measure the performance of regression by 5-fold cross validation on the generated training set. The model is trained on 4 subsets out of 5, partitioned uniformly, and tested on the remaining one. We repeat this experiment 5 times, and by taking its average, we get the average R^2 value of the system.

We show the numerical predicted value R^2 for all clouds in Table 4.4. More the R^2 score, more correlated the estimated value is to the actual value. The accuracy of the assignment problem depends on the accuracy of the predicted value by the statistical regression model. Here we see, in all cases, the avg R^2 value is greater than 0.75 which implies it is possible to predict the value of all the measurements and time with higher accuracy, hence validating the use of statistical regression in our model.

Prediction metric	**Avg R^2 score**
CPU Performance	0.770162381924
Network Bandwidth	0.937836016201
Local Memory Bandwidth	0.877783301925
Throughput	0.891278955663
Latency	0.749202490590
Time	0.928851419281

Table 4.4: Avg R^2 score for aggregated prediction measurement

4.4.4 Cost Comparison

Once the Classifier model is up and working, we estimate both the classes for a new workload i.e. the test data. For cost minimization, we apply the above mentioned technique and present the experiments done comparing the base prices of Amazon Web Services, Microsoft Azure and Google Compute Engine with the estimated prices of CM(EM) after applying all the discount model and schemes. We have conducted experiments from 20th Feb'2017 to 26th Feb'2017, because, we need to consider the factor that on different days and on different times, there are different discount schemes for all public cloud provider. From the graphs, we see the average discount(%) provided by our framework for each day. "Also, we take the average discount aggregating all the days of the week, which results into an ***avg. 65.1345%*** discount applied on any user submitted workload compared to paying the on-demand price

for the particular resource needed. The other inference to be made by seeing the graphs are, that the system performs very well on GPU based instances, where the cost of the resources are itself quite high and so by selecting the right set of instances along with applying the discount model, we could significantly lower the prices. We get an average discount of more than ***75%*** for 90th percentile and more, which is quite noteworthy." We now show Test workload data generated and evaluated for a week (vs price to run a workload) from *20th Feb'2017* to *26th Feb'2017*

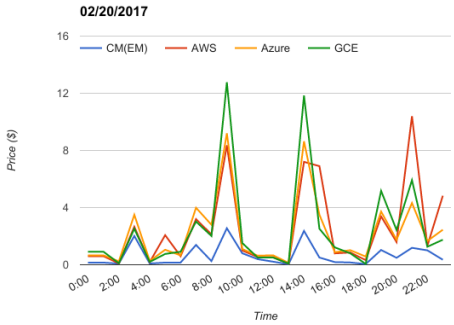

Figure 4.2: 60.696% avg. discount on 20th

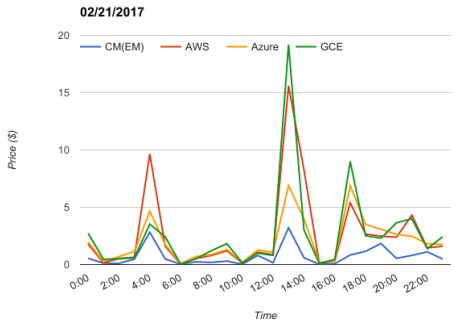

Figure 4.3: 59.646% avg. discount on 21st

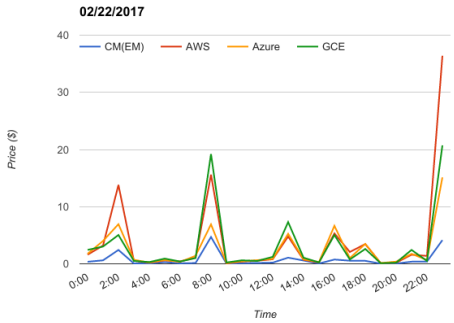

Figure 4.4: 70.9225% avg. discount on 22nd

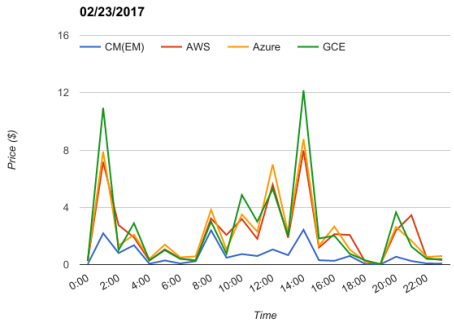

Figure 4.5: 64.491% avg. discount on 23rd

4.5 Miscellaneous System Features

Periodic updates of changing costs

There are constantly changing prices online from where we extract our data. Our architecture runs a cron job (automated scheduled job without manual intervention needed) every hour to update the prices i.e the key-value pair stored in the map of Cost Calculator. Also, EC2 provides a command line API to see the spot history from your preferred start and end time, we use it for predicting/forecasting the spot bid on Amazon Web Services.

Termination of instances

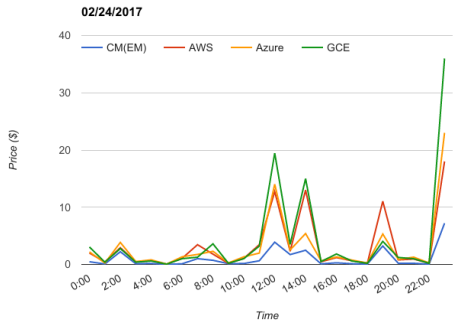

Figure 4.6: 67.597% avg. discount on 24th

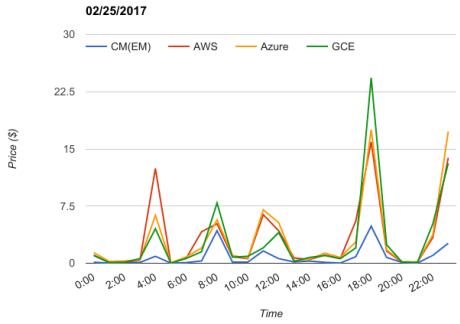

Figure 4.7: 66.142% avg. discount on 25th

As we know preemptible instances of GCP and spot instances in AWS are sensitive to termination. So, sometime the user want the log and the data or the results of the workloads, but the instances are terminated. "CM(EM)" handles this by landing the process in a safe state where the user has all the data and the logs before the instance is terminated. We do this by giving notification/alerts when the market price gets greater than yours, or when your instance is going to be terminated. Immediately after this, we start calculating the price again and start running the workload on a different instance of the chosen public cloud provider.

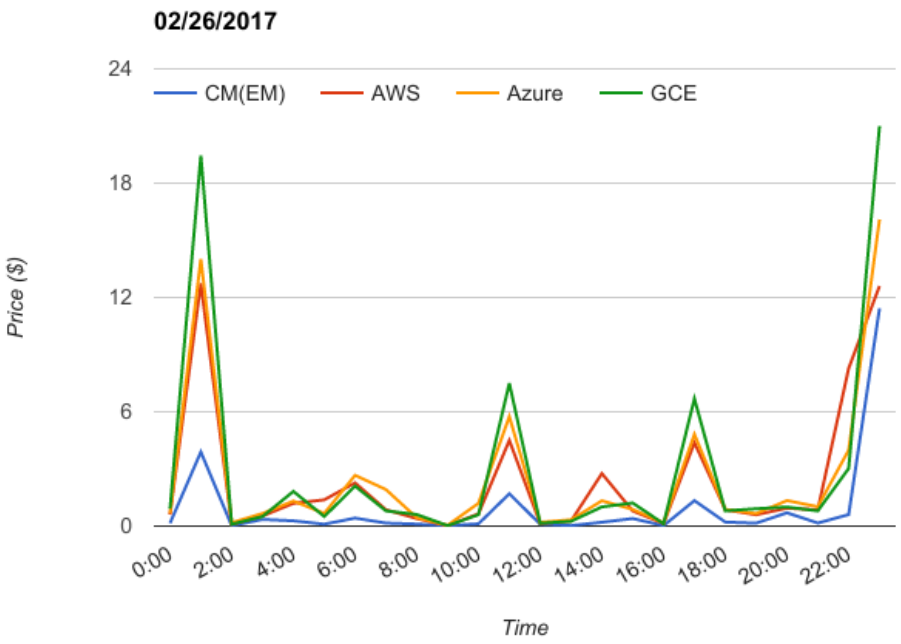

Figure 4.8: 66.447% avg. discount on 26th

4.6 Summary

The problem of providing optimal assignment for IO workload with respect to compute and storage was a central problem in the design of multi-cloud heterogenous systems. So, in this chapter, we have tried to achieve and represent "optimality" by designing a pipeline model which selects a compute and a storage class for a given workload for efficiency maximization and then apply discount schemes provided by the public cloud providers for cost minimizations. We also used some parameters provided by the user inputs to fine tune the cost minimization procedure. We have designed these methods and algorithms using these three public cloud providers: Amazon Web Services, Microsoft Azure and Google Cloud Platform.

Chapter 5

Future Work and Conclusions

In this final chapter, I could conclude my thesis and explain some of the future work planned ahead.

5.1 MultiStack

MultiStack is under development and there are lot of interesting features being added. In this section we brief about the work currently in progress as part of the MultiStack project. We have detailed MultiStack, a cost and deadline aware middleware for running big data processing jobs across multiple clouds.

Dynamic Replication in Storage Systems

With the Algorithm mentioned in the 2nd Chapter, we can scale processing of data clusters at low cost, but storage system can become new bottleneck once the processing is scaled. We are exploring techniques to identify access patterns of storage objects and use them to optimize the replica count of objects and their placement. Frequency based load balancing is traditionally applied in such scenarios [30] [39], but we are working on techniques which go beyond simple frequency based approaches and uses other high level features like user modeling as well. We are in stage of combining learned user models with learned data models into a single unified model.

Fault Prediction

Use of device data and past faults to predict future faults. This work is in early phase and we have not collected enough fault data to conduct useful experiments.

Interplay of various learning sub-system

With more than one learning based models in a system, we want to characterize the interplay among them. The decisions of different learning systems are interlinked (coherent system decision).

Auto-selection of data processing frameworks

It is well known that not all frameworks are made equal. End-user of the data processing system often does not understand the design decisions behind data processing framework designs. They might not know the right tool for the job. Choosing right framework is not a straight forward procedure given that it depends on complex data points such as data size, computation nature etc.

We propose to combine learning and heuristics based methods to identify most suitable platform for running a given processing task for the given data. A graph showing performance of a algorithm on different framework and different data size.

5.2 HetStore

We have focussed on the utility of regression models for estimating the performance of workloads in a heterogeneous SDS environment, in order to improve aggregate performance. However, the measure of this aggregate performance depends on the particular needs and requirements in the respective design or engineering problem. The choice of the cost function, therefore, would largely depend upon the specific problem at hand. The use of this system towards a general class of cost functions has been left open. It could be fruitful to identify some relevant classes of cost function, which are also amenable to analytic scrutiny (eg. linear cost functions over throughput and IOPS), and look at the performance of SVR-based classification over these classes.

The architecture is also extendible, and with only minor modifications can also incorporate metrics other than throughput and IOPS. We believe a better selection of features could also possibly provide better accuracy and remains to be explored. For instance, while we have not included performance metrics like latency in our system, similar estimation mechanism for latency could also be incorporated.

Also, as we note, some of the limitations of our system arise from the dynamic nature of the underlying SDS as well as the misalignment of distributions from which training data is sampled. In order to overcome that, an Online approach to this learning problem might provide better results and the challenge remains in finding such an approach without excessive overhead computation.

The limited availability of data sets on workloads result in a lack of available insight on the prevalent statistical distributions of workloads used in industry and practice. While online approach will resolve the problem for a specific use-case of this system, availability of standardized data sets would allow better benchmarking for such methods.

5.3 CM(EM)

We apply cost minimization after selecting/predicting a network efficient environment for this multi-cloud heterogenous setup. But here we have not considered hybrid scenarios with instances and storages of different public cloud running a distributed workload/job. We see a lot of scope in improving the assignment through prediction by statistical modelling where we could use hybrid resources for a workload using more instances. We plan to continue investigating in a scenario where we could use this hybrid architecture and continue to learn about some heuristics in multi-cloud interoperability.

There are features which we did not consider while designing the pricing model for Amazon Web Services and Google Cloud Platform. In AWS, we ignored the Reserve Instances (RI), which are basically 1 or 3 year contract providing a discount upto 75% where you could schedule instances with day-date-time basis. We need regular traffic in our system to take its benefit and due to lack of user data for this architecture, we are not able to include this in our experiments as well as evaluations. Also, in AWS, there is a new feature added called Auto Scaling Group for spot instances, which would automatically match the market's bid and would launch a new replacement instance. In Google Cloud Platform, it is permissible to make our own custom instances, so you could make your own optimal instance with the resources and configuration of your own choice. GCP provides a Google Cloud Platform Pricing Calculator[11], which would calculate the price of your custom instance given the configs, for this we need to apply resource prediction and optimal configuration assignment for a given IO workload. Also, we did not perform experiments on all machine types like Microsoft distro and storage types like File Storage, so we would try to extend these experiments by adding these parameter.

We also note that there are some limitations in our model due to the dynamic nature of our architecture and we assume that it is a fault tolerant system. The user could lose some data if the instance is terminated or due to quota failures. We also would like to explore some online approach to this prediction model which might provide better results without excess overhead.

Also, limited availability of dataset of workload also prevents us from gaining insight on types and classes of workload distribution. We are investigating different statistical model application using online algorithms and so hope to make some improvements in both network and cost optimizations.

Bibliography

[1] Amazon aws. https://aws.amazon.com/.

[2] Amazon elastic compute cloud (ec2). http://aws.amazon.com/ec2.

[3] Amazon elastic mapreduce (amazon emr). http://aws.amazon.com/elasticmapreduce/.

[4] Apache hadoop. http://hadoop.apache.org/.

[5] Apache oozie. http://oozie.apache.org/.

[6] Ceph. https://ceph.com/.

[7] Cloudstack. https://cloudstack.apache.org/.

[8] Emc vipr. http://www.emc.com/products/storage/software-defined-storage/vipr-controller.htm.

[9] Fio - flexible io tester. https://github.com/axboe/fio/.

[10] Glusterfs. https://www.gluster.org/.

[11] Google cloud platform pricing calculator. https://cloud.google.com/products/calculator/.

[12] Google compute engine. https://cloud.google.com/compute/.

[13] Hadoop on cloud. http://siel-iiith.github.io/MultiStack/.

[14] Hdfs. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[15] Ibm virtual storage. http://www-03.ibm.com/software/products/en/ibm-virtual-storage-center.

[16] iperf. https://iperf.fr/.

[17] Microsoft azure. https://azure.microsoft.com/en-in/.

[18] Netapp ontap. http://www.netapp.com/us/products/platform-os/ontap/.

[19] Openstack. https://www.openstack.org/.

[20] Openstack swift. http://docs.openstack.org/developer/swift/.

[21] Project serengeti. http://www.projectserengeti.org/.

[22] Support vector machine. https://en.wikipedia.org/wiki/Support_vector_machine/.

[23] A. Alba, G. Alatorre, C. Bolik, A. Corrao, T. Clark, S. Gopisetty, R. Haas, R. I. Kat, B. Langston, N. Mandagere, et al. Efficient and agile storage management in software defined environments. *IBM Journal of Research and Development*, 58(2/3):5–1, 2014.

[24] C. Albrecht, A. Merchant, M. Stokely, M. Waliji, F. Labelle, N. Coehlo, X. Shi, and E. Schrock. Janus: Optimal flash provisioning for cloud storage workloads. In *USENIX Annual Technical Conference*, pages 91–102, 2013.

[25] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. N. Tantawi, and C. Krintz. See spot run: Using spot instances for mapreduce workflows. *HotCloud*, 10:7–7, 2010.

[26] A. Ganapathi. Predicting and optimizing system utilization and performance via statistical machine learning. 2009.

[27] C.-H. Ho and C.-J. Lin. Large-scale linear support vector regression. *Journal of Machine Learning Research*, 13(Nov):3323–3348, 2012.

[28] J. L. Horey, E. Begoli, R. Gunasekaran, S.-H. Lim, and J. J. Nutaro. Big data platforms as a service: Challenges and approach. In *HotCloud*, 2012.

[29] B. Javadi, R. K. Thulasiramy, and R. Buyya. Statistical modeling of spot instance prices in public cloud environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 219–228. IEEE, 2011.

[30] K. Jindarak and P. Uthayopas. Enhancing cloud object storage performance using dynamic replication approach. In *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pages 800–803. IEEE, 2012.

[31] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda. Characterization of storage workload traces from production windows servers. In *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pages 119–128. IEEE, 2008.

[32] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[33] N. J. Nagelkerke. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, 1991.

[34] B. Seo, S. Kang, J. Choi, J. Cha, Y. Won, and S. Yoon. Io workload characterization revisited: A data-mining approach. *IEEE Transactions on Computers*, 63(12):3026–3038, 2014.

[35] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1013–1020. ACM, 2010.

[36] B. Tremblay, K. Kozubal, W. Li, and C. Padala. A workload aware storage platform for large scale computing environments: Challenges and proposed directions. In *Proceedings of the ACM 7th Workshop on Scientific Cloud Computing*, pages 27–33. ACM, 2016.

[37] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. *Future Generation Computer Systems*, 29(4):973–985, 2013.

[38] D. van der Ster and A. Wiebalck. Building an organic block storage service at cern with ceph. In *Journal of Physics: Conference Series*, volume 513, page 042047. IOP Publishing, 2014.

[39] Q. Wei, B. Veeravalli, and Z. Li. Dynamic replication management for object-based storage system. In *Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on*, pages 412–419. IEEE, 2010.

[40] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 122. ACM, 2006.