ENERGY EFFICIENT DYNAMIC VIRTUAL MACHINE ALLOCATION WITH CPU USAGE PREDICTION IN CLOUD DATACENTERS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

By

Gökalp Urul

January 2018

ENERGY EFFICIENT DYNAMIC VIRTUAL MACHINE ALLOCATION WITH CPU USAGE PREDICTION IN CLOUD DATACENTERS

By Gökalp Urul

January 2018

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

İbrahim Körpeoğlu(Advisor)

Özgür Ulusoy

Ahmet Coşar

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

ENERGY EFFICIENT DYNAMIC VIRTUAL MACHINE ALLOCATION WITH CPU USAGE PREDICTION IN CLOUD DATACENTERS

Gökalp Urul
M.S. in Computer Engineering
Advisor: İbrahim Körpeoğlu
January 2018

With tremendous increase in Internet capacity and services, the demand for cloud computing has also grown enormously. This enormous demand for cloud based data storage and processing forces cloud providers to optimize their platforms and facilities. Reducing energy consumption while maintaining service level agreements (SLAs) is one of the most important issues in this optimization effort. Dynamic virtual machine allocation and migration is one of the techniques to achieve this goal. This technique requires constant measurement and prediction of usage of machine resources to trigger migrations at right times. In this thesis, we present a dynamic virtual machine allocation and migration method utilizing CPU usage prediction to improve energy efficiency while maintaining agreed quality of service levels in cloud datacenters. Our proposed method, called LRAPS, tries to estimate short-term CPU utilization of hosts based on their utilization history. This estimation is then used to detect overloaded and underloaded hosts as part of live migration process. If a host is overloaded, some of the VMs running on that host are migrated to other hosts to avoid SLA violations; if a host is underloaded, all of the VMs in that host are tried to be migrated to other machines so that the host can be powered off. We did extensive simulation experiments using CloudSim to evaluate the efficiency and effectiveness of our proposed method. Our simulation experiments show that our method is feasible to apply and can significantly reduce power consumption and SLA violations in cloud systems.

Keywords: cloud, virtual machine, resource allocation, local regression, cross validation, dynamic allocation.

ÖZET

İŞLEMCİ KULLANIM TAHMİNİYLE ENERJİ VERİMLİ DİNAMİK SANAL MAKİNE YERLEŞTİRMESİ

Gökalp Urul
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: İbrahim Körpeoğlu
Ocak 2018

İnternet kapasitesinin ve servislerinin son zamanlardaki hızlı artışı bulut bilişim ihtiyacının da hızla artmasına sebep olmaktadır. Bulut temelli bilgi depolama ve işlemci gereksinimi bulut sağlayıcıların kendi platformlarını ve tesislerini en iyi şekilde kullanmalarını zorunlu hale getirmektedir. Tesislerin ve ekipmanların en iyi şekilde kullanılabilmesi için enerji tüketiminin azaltılması ve servis anlaşmalarının belirli bir seviyede tutulması gerekmektedir. Dinamik sanal makine yerleşmesi ve taşınması bu hedefe ulaşmada kullanılan yöntemlerden biridir. Bu yöntemin en iyi şekilde uygulanabilmesi için fiziksel makine kaynak kullanım oranlarının sürekli gözlemlenmesi ve tahmin edilmesi gerekmektedir. Bu tez çalışmasında, fiziksel makine işlemci kullanım oranlarını tahmin ederek enerji tasarrufu ve servis kalitesi sağlayan yeni bir dinamik sanal makine yerleştirme ve taşınma yöntemi sunmaktayız. Önerdiğimiz yöntem, LRAPS, fiziksel makinelerin geçmiş işlemci kullanım bilgilerine bakarak gelecekte gerekecek işlemci kullanım oranını tahmin etmeye çalışmaktadır. Daha sonra bu tahmin kullanılarak normal yükleme değerinin üstünde ya da altında olan fiziksel makineler tespit edilmektedir. Eğer bir makine aşırı yüklenmişse, o makinede bulunan bazı sanal makineler servis anlaşmasını bozmamak için uygun olan diğer fiziksel makinelere taşınır; eğer bir makine az yüklenmişse, o makinede bulunan bütün sanal makineler enerji tasarrufu sağlamak için uygun olan diğer fiziksel makinelere taşınır. Yöntemimizin performasını ve etkinliğini görmek için kapsamlı simülasyon deneyleri gerçekleştirdik. Simülasyon deneyleri yöntemimizin uygulanabilir olduğunu, enerji tasarrufu yapmanın yanı sıra servis anlaşmasını sağladığını da gösterdi.

Anahtar sözcükler: bulut bilişim, sanal makine, kaynak ayırma, yerel regresyon, çapraz onaylama, dinamik yerleştirme.

Acknowledgement

I would like to take this opportunity to express my gratitude to my supervisor, Prof. Dr. İbrahim Körpeoğlu, for his motivation, guidance, encouragement and support throughout my studies.

I would also like to thank to Prof. Dr. Özgür Ulusoy and Prof. Dr. Ahmet Coşar for kindly accepting to spend their valuable time to review and evaluate my thesis.

I express my graditute to my brother Gökhan Urul, my mother Mücella Urul, my father Mehmet Urul and İrem Anter for always being motivating and supportive. None of this would have been possible without their love, selflessness and sacrifices they made on my behalf.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Virtualization is one of the most fundamental technologies of cloud computing, which enables creation of multiple virtual machines to run simultaneously on the same physical server. With the growing demand for cloud computing, use of virtualization technologies has become a must for cloud providers for offering various cloud services to users in a flexible manner. Cloud services can be categorized into three major types: SaaS, PaaS and IaaS. Commercial IaaS providers, like Amazon [1], offer different types of virtual machines to users, which can be used flexibly via the pay-as-you-go charging model.

Demand for cloud computing resources and virtual machines by users, however, has a dynamic nature and varies over time, and therefore the exact demand at a future point in time is not exactly known. This forces cloud providers to optimize their datacenters with dynamic virtual machine allocation and migration. Otherwise, if a cloud provider does not efficiently allocate virtual machines to physical servers to match changing demands, some hosts may become overloaded which may cause SLA violations, or may become underloaded, which may cause unnecessary energy consumption. Both situations cost a lot of money to cloud providers.

In order to efficiently and dynamically place virtual machines into physical

servers, a technique called live virtual machine (VM) migration is usually used. Live VM migration is a powerful feature of virtualization since it allows dynamic consolidation of virtual machines to a minimum number of physical servers. When a host is overloaded, some of the VMs can migrate to other available physical servers without stopping the running applications and degrading QoS (quality of service). Or if a host is underloaded, all of its VMs can migrate to other hosts so that the underloaded host can be put into sleep mode or powered off to save energy.

In cloud computing, QoS is formalized in terms of SLAs (service level agreements) between cloud providers and users. An SLA is a policy governing the use of a cloud service under the terms of the provider. The terms may change for different providers, however, the main idea is to promise the end user that there will not be any failure or service degradation beyond a certain agreed threshold rate. In case the realized service level does not comply with the agreement, the cloud provider may have to give credits to the end user.

During dynamic virtual machine allocation, providers need to maintain QoS while reducing energy consumption, since there is usually a trade-off between them. For instance, if a physical server is not fully loaded, virtual machines running on it can get any resources they need, even if the resource demand suddenly increases. These extra available resources provided to VMs on that host may result in fewer SLA violations, however, since the server is not adequately utilized, this will cause unnecessary energy consumption. Therefore, live VM migration process should not jeopordize quality of service while trying to consolidate virtual machines to save energy.

Dynamic VM allocation process mainly consists of the following four parts:

- Determining an overloaded host,
- Determining an underloaded host,
- Determining which VMs need to be migrated from an overloaded host or from an underloaded host,

- Finding and selecting suitable destination hosts to where migrations of VMs will take place.

The process will be executed for each overloaded or underloaded host.

In this thesis we are focusing on first two operations of dynamic VM allocation process. Detecting overloaded and underloaded hosts accurately has enormous impact on overall allocation process, since it triggers the live migration which is a quite costly operation. If the detection is wrong, an unnecessary migration can occur or a necessary migration can be missed.

There are different resource types of a server, which can be checked for overload and underload conditions, like CPU, bandwidth, I/O and memory. In this thesis we are focusing on CPU usage, since it is usually the most energy consuming part in servers. Predicting CPU usage accurately can help making timely decisions to migrate virtual machines between servers.

In this thesis we propose a regression based CPU usage prediction algorithm, called LRAPS (Local Regression Automated Parameter Selection), to predict hosts' future CPU utilizations and make sound migration decisions based on these predictions. Our LRAPS algorithm uses local regression method which is a non-parametric approach that fits multiple regressions in local neighborhood. However, during local regression, instead of selecting a static span (a smoothing parameter), we are using GCV (general cross validation) method to find the best span value for the current dataset, which results in better predictions. Hence we are using a dynamic span as part of our prediction algorithm.

Our work shows that predicting hosts' future CPU utilization provides useful information during the detection process. With this prediction we can detect overloaded or underloaded hosts before the problem occurs so that we can take action in advance.

We conducted extensive simulation experiments to investigate the performance

of our LRAPS algorithm. First we analyzed the effect of dynamic smoothing parameter selection on local regression prediction. It shows that optimal smoothing parameter varies with datasets, so using a static span decreases the accuracy of the prediction. By using generalized cross validation (GCV) method we found the optimal span value and used that in local regression prediction. After the analysis, we implemented our LRAPS algorithm into VM allocation and migration process. Simulation results show that during overloaded or underloaded host detection, using future CPU utilization rather than current CPU utilization prevents unnecessary migrations. We also compared the regular local regression with our LRAPS algorithm and observed that using a dynamic smoothing parameter instead of a static parameter improves the performance of prediction significantly.

The contributions of the thesis can be summarized as follows. 1) We analyze different methods to find the optimum smoothing parameter for local regression. 2) We propose a novel approach to detect overloaded and underloaded hosts during dynamic VM allocation process. 3) We extend CloudSim simulation environment to support our prediction algorithm. 4) We compare our approach with other proposed methods via real-world and computer generated datasets. 5) As the final contribution, we show that our algorithm improves the efficiency of virtual machine allocation in large datacenters.

The remainder of the thesis is organized as follows. In Chapter 2 we discuss different virtual machine allocation and host selection methodologies. We also give some background information about virtualization and VM allocation. In Chapter 3 we present our prediction algorithm and its implementation together with the related mathematical models. In Chapter 4, we explain our simulation environment, metrics and experimental setup. Then experimental results are given and discussed. Finally, in Chapter 5, we give our conclusions and thoughts about possible future works.

Chapter 2

Related Work and Background

In this chapter, we will discuss the previous works on the topic of live migration and overloaded and underloaded host detection. We will also give some background information about virtualization technologies, virtual machine allocation strategies and live migration techniques.

2.1 Related Work

In [2], Bleaglazov and R. Buyya proposed a static threshold method for host selection. In their study they suggested a lower and an upper threshold for CPU utilization. If the current CPU utilization of a host is not between these two thresholds then a live migration process occurs. If the current CPU utilization exceeds upper threshold the host is labeled as overloaded and some of the VMs on it are migrated to other available hosts to maintain QoS. On the other hand, if the current CPU utilization is below lower threshold the host is labeled as underloaded and all of the VMs in it are migrated to other available hosts to put host into sleep mode. By doing that the number of active servers are reduced. This methodology is very simple and doesn't require any additional process, however, it is not suitable for dynamic environment since workloads are constantly

changing.

In [3], authors proposed four different overload condition detection algorithms which can be considered in two different categories. First category is adaptive utilization threshold. In this category they used MAD (median absolute deviation) and IQR (Inter Quantile Range) algorithms. Instead of setting a static threshold like in [2], these algorithms are adjusting their utilization threshold automatically by using historical data of hosts and their strengths of deviation. Second category is regression based utilization threshold. This category includes two algorithms which are Local Regression (LR) and Robust Local Regression (RLR). The main idea of the method of local regression is fitting simple models to localized subsets of data to build up a curve that approximates the original data. In regression based methods they are trying to predict the next CPU demand of the host so that they can migrate virtual machines before any overloading or underloading occurs. However, in order to make their local regression algorithm responsive enough to the latest values, they are using a static span value for all predictions which results in bad approximations.

In [4], researchers proposed a linear regression algorithm to predict the next CPU utilization of the host (LIRCUP). The main idea of this algorithm is to use historical data to find a function that fits the best and use this function to predict the next CPU usage. Since it is not always possible to find a fitting function because of the chaotic nature of CPU demand, sometimes it results in weaker predictions.

K. Maurya and Shina proposed adaptive migration threshold algorithm in their study [5]. Their algorithms are not using historical data, instead they are using current utilization to adapt the upper and lower thresholds dynamically. Also they implement their algorithms for three dimensions which are CPU, RAM and bandwidth. Since they just use the current utilization, some SLA violations cannot be foreseen. If there is an unpredicted short spike, this solution may label the server as overloaded which is unnecessary.

In [6], authors proposed averaging threshold algorithm for detection. Similar

to [2] the upper and lower thresholds are fixed to a certain value, however, instead of using current utilization they are using the mean of the past n CPU utilization samples. If the average of n past CPU utilization samples exceeds the upper threshold, the host is labeled as overloaded. When compared to [2] this averaging method gives better results. However, like all static threshold methods it is not suitable for dynamic workloads.

In [7], authors proposed Fuzzy Q-Learning based detection algorithm. In their study, a fuzzy clustering algorithm is used to find the Gaussian functions. Like adaptive threshold, for every host new threshold is selected according to the feedback. One of the drawbacks of this algorithm is that the convergence time is too long. Therefore for the real time applications convergence time can cause problems.

Authors in [8] also proposed a method which dynamically adapts the upper and lower thresholds. Different from the rest they use both predicted and current CPU utilization values in their algorithm. After collecting all the related data of a host, they put these information to fuzzy inference engine. This engine is supported by the Sugeno fuzzy rule. Also, workload changes are measured continuously for the lower bound.

In [9], authors proposed a virtual machine based dynamic threshold. Their algorithm is used to detect overloaded and underloaded hosts by looking at the number of virtual machines and CPU utilization of the hosts. This algorithm also uses the current utilization data instead of historical data. Hence, it is not very adaptable for dynamic environments.

In [10], Ching-Hsien Hsu et al. proposed an energy-aware task consolidation algorithm (ETC) to minimize the energy consumption. They are minimizing the energy consumption by restricting CPU use below a specified peak threshold by migrating and consolidating tasks amongst virtual clusters and virtual machines. Their model also considers the network latency during migration. However, keeping the CPU usage between two static thresholds is not suitable for dynamic environments.

In [11], authors proposed a system to allocate datacenter resources dynamically based on application demands. They used exponentially moving average method to predict the future CPU usage. To decide hot spots they set different thresholds for different resource types.

In [12], Sadeka Islam et al. proposed two learning algorithms to predict future resource demands. They used linear regression and neural networks for prediction. Their experiments show that neural network model gives better prediction accuracy than linear regression. However, the training process of neural network model takes significant amount time depending on the size of the input as well as the frequency of the prediction. Also time series of workloads may be chaotic which does not depend on the previous patterns.

In [13], authors are predicting the number of VMs, also their CPU and memory demands. By using these predictions they are allocating the VMs to physical machines efficiently. If there are unnecessary physical machines running up, they put them into sleep mode. However, virtual machine migration is very important for continuous allocation, which means that the overload condition detection should be done for all the time. Load monitoring at later times is even more important than the VM submission time. Since they are not focusing on the continuous allocation, their solution is not very suitable for dynamic environments.

Fahimeh at al. [14] proposed a dynamic virtual machine consolidation algorithm based on k nearest neighbor regression to predict resource usage in each host. In KNN algorithm they used past m CPU usage data to predict the demand at time $m + 1$. They used this prediction to decide whether the host is overutilized or underutilized. They minimize the energy consumption and SLA violation rate more effectively than other dynamic consolidation methods.

The study conducted in [15] presents an approach, called statistic based load balance (SLB), to make online resource allocation decisions. They are using historical data to predict the VM's resource demand for the future. However, instead of using a regression or machine learning model for their prediction, they simply calculate the average of the past CPU usage of servers and used that

Methods	**Live Migration Host Selection Parameters**						**Evaluation Metrics**		
	Number of Vms on the host	Historical Data	Current CPU Utilization	Static Threshold	Dynamic Threshold	Future CPU Utilization	Power Consumption	SLA Violation	Number of Migration
Belaglazov et al. [2]			✓	✓			✓	✓	✓
THR [6]		✓		✓			✓	✓	✓
MAD [3]		✓			✓		✓	✓	✓
IQR [3]		✓			✓		✓	✓	✓
LR [3]		✓		✓		✓	✓	✓	✓
RLR [3]		✓		✓		✓	✓	✓	✓
LirCUP [4]		✓		✓		✓	✓	✓	✓
Adaptive Migration Threshold [5]			✓		✓		✓	✓	✓
DFQL [7]	✓	✓			✓		✓	✓	✓
Adaptive Fuzzy Threshold [8]		✓			✓		✓	✓	✓
VDT [9]	✓		✓		✓		✓	✓	✓
EDT [10]		✓		✓		✓	✓	✓	✓
Exponential Moving Average [11]		✓		✓		✓	✓	✓	✓
Neural Network [12]		✓		✓		✓	✓	✓	✓
kNN [14]		✓		✓		✓	✓	✓	✓
SLB [15]		✓		✓		✓	✓	✓	✓
LRAPS		✓		✓		✓	✓	✓	✓

Table 2.1: Comparison of Host Overload Detection Algorithms.

average as the future CPU demand.

In the Table 2.1 you can find the summary of the related works. For every host detection algorithm mentioned in the paper we showed which parameters and which evaluation metrics are used. This chart helps us to see the differences between the algorithms in a clear way. Our LRAPS algorithms is located at the bottom of the cart.

2.2 Virtualization

Virtualization is one the most fundamental technology used in cloud and dedicated servers. Today's datacenters consist of computers with high cores, more storage and more memory. Unlike traditional computers, these high power computers are preferred in deployment because buying more computers with smaller resources is too expensive and not feasible for the cloud providers. Even though powerful computers are more feasible than the smaller ones, serving a powerful computer to one client may not be desired all the time, because one client's resource demand can be much lower than what is provided. Hence leasing all the resources to one client would not be cost effective from the customer side. In order to efficiently allocate resources to many users, virtualization technique is used. Virtualization enables to create several different virtual machines inside a physical machine so that different users can benefit from a high power computer at the same time.

In datacenters server virtualization is used to create virtual machines. A key to server virtualization is a software layer called hypervisor. Hypervisor isolates the operating system from the underlying computer hardware like CPU, memory, network and I/O. There are two types of hypervisors as seen in Figure 2.1; Type 1 hypervisors sit directly on bare metal and does not require a host operating system, such as vSphere [16] or Hyper-V [17]. This type of hypervisors are popular for the cloud providers. Type 2 hypervisors run as a software layer on an operating system, such as VMware Player [18] or Parallels Desktops [19].

Type 1 hypervisors, used in cloud, abstract the guest operating system from the hardware of the physical machine without using a host operating system. Such abstraction enables guest operating system, normally interacting with the actual hardware, can be run on the emulated hardware. Even most of the time the guest operating system does not know it is on an emulated hardware. The difference between the traditional architecture and virtual architecture is shown in Figure 2.2

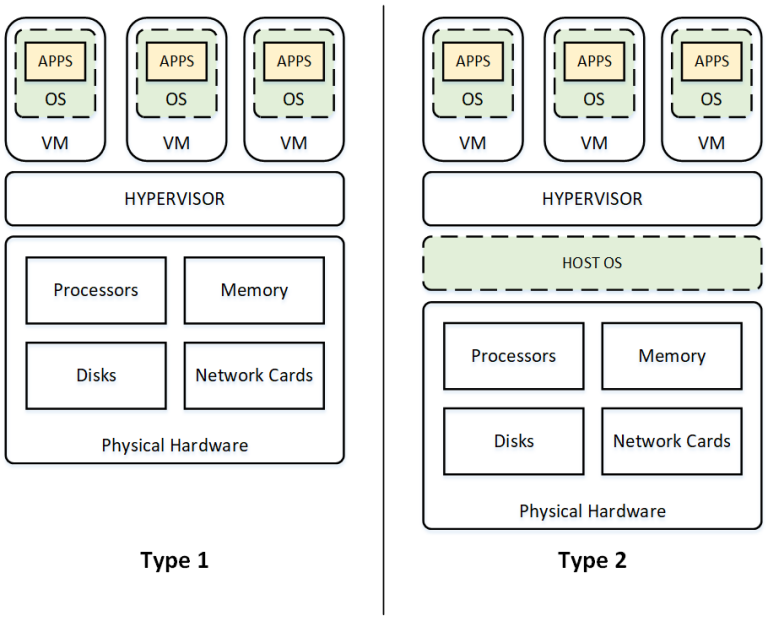


Figure 2.1: Hypervisor Types.

Even though the performance of these guest operating systems has some deficiencies, allocating physical resources to several virtual machines is way more efficient than the traditional architecture for cloud providers and users.

Hypervisors provide lots of benefits to cloud providers. Like we mentioned before, they enable physical computers to run several different VMs simultaneously on the same machine. Also since hypervisors make the virtual machines independent from the underlying hardware, VMs can be moved from one physical machine to another easily. Traditional software is highly coupled on the server hardware which causes time consuming migration process. However, hypervisors decreases the time of moving processes tremendously. Such VM migration enables cloud providers to consolidate their VMs to minimum number of physical machines, which we focus on in this thesis.

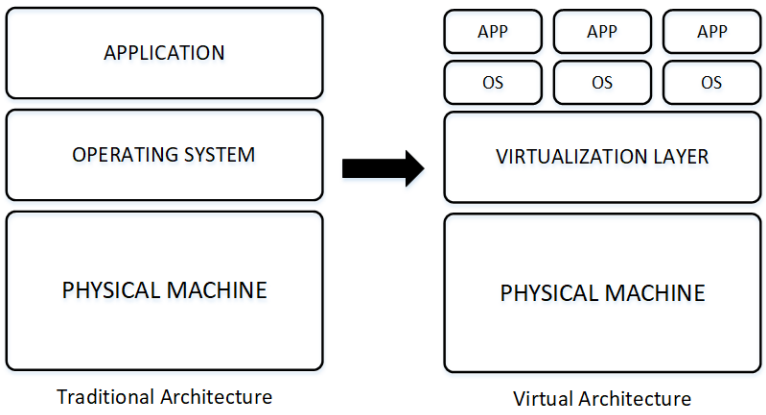


Figure 2.2: Traditional Architecture vs Virtual Architecture.

2.3 Virtual Machine Migration

VM migration is the process of moving a virtual machine from one host to another host without stopping or disconnecting the applications running on it. There are several important use cases of virtual machine migration such as load balancing, maintenance and recovery from host failure. Since the number of hosts in a datacenter vary up to thousands, the requirement of maintenance and recovery from host failure is inevitable. In such cases the IT team of the cloud provider can migrate virtual machines to other available hosts before the problematic host is shut down.

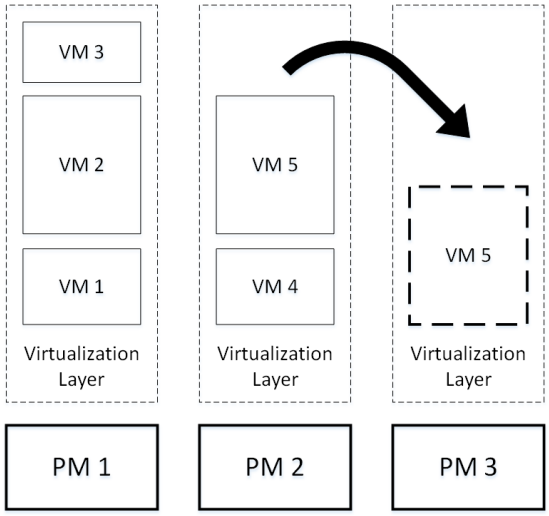


Figure 2.3: Live VM Migration.

Besides maintenance and recovery, especially load balancing may require higher number of VM migrations. For instance, if one of the VMs on a host start to demand more CPU than the host can provide, this may cause overutilization which may result in SLA violation. Or on the contrary, if the resource demand of a VM on the host is too little, this may cause underutilization which results in unnecessary energy consumption. Cloud providers try to utilize physical machines by constantly migrating VMs between hosts. During VM migration it is also important that the connectivity of the user or application should be maintained such that the migration process will not be noticeable from the user point of view.

There are several different migration types such as cold, warm and live migration. In cold migration VM that will be moved is shutdown on the host, then all the content is copied to the destination host. After that, the same VM is restarted. In warm migration VM is temporarily suspended. Registers of the VM are copied to the destination host. After that, VM continues to run on the destination host. Live migration, third type, is the most important one in cloud environment since it has the least effect on the user connectivity.

The main purpose of live migration is to minimize the down time and total migration time of a VM during the moving process. If the down time of a VM is not noticeable by the end user then it is called seamless live migration. During live migration the VM continues to run while the RAM is copied to destination host (Figure 2.3).

During live migration, transferring memory state of the virtual machines is very important. There are two different techniques used in the cloud computing to move virtual machine's memory state from one host to the other, which are pre-copy memory integration and post-copy memory integration. Even though these two techniques are different from each other, actually both of them are based on three main memory transfer phases: push phase, stop and copy phase, and pull phase [20].

In push phase while a VM is running on the source host, most of the state pages are sent to the destination host. However, since the VM is still running,

some pages are modified on the source during this time, which means that some of the pages sent to destination host are not same as the source. In order provide consistency, the pages modified during the transmission process are resent to the destination.

Stop and copy phase is the one where the source VM is stopped to send the modified memory pages to the destination. After the VM is stopped, all the memory pages are copied across the destination. After the transmission is done, VM in the destination host is started.

In the pull phase the migrated VM starts its execution at the destination host, however, the source VM is still present and may provide some supporting in case the new VM may need to access some pages that has not been copied yet. For instance, until the network fabric has caught up with the new location of the target VM, the source VM provides forwarding services for packets to and from the target VM.

Pre-copy migration starts with the push state. Generally hypervisor copies all the memory pages to the destination host without disrupting the VM running on the source. If some pages are modified during copy process, they are recopied. After push state, the VM on the source is stopped and the remaining modified pages are sent to destination VM and the VM is resumed on the destination host. The time between when the VM is shutdown on the source and the new VM is started on the destination is called as down-time. Down time can vary from few milliseconds to seconds according to memory size and the application.

Post-copy memory integration is different than the pre-copy. VM migration is started by suspending the VM on the source. Then some of the important execution states such as CPU state, registers and non pageable memory is sent to the destination host. Since these states are enough for execution, the VM continues on the target host without waiting for the rest of the state pages. At the same time remaining memory pages are copied to the destination host. If the VM on the destination tries to reach a page that has not been copied yet, it creates a error. These errors are captured at the destination and redirected

to the source. When the source gets the error information, it resends the pages which has errors.

Unlike pre-copy migration, post-copy migration sends each page only once during the migration. If there are dirty pages at the source, pre-copy needs to resend these pages to the destination. On the other hand, during the migration if an error occurs and the destination VM fails, it is not possible to recover VM with post-copy migration, since there is no up-to-date state of the VM. However, pre-copy migration keeps up-to-date state of the VM at the source during migration. Timeline of two different migration techniques is shown in the Figure 2.4.

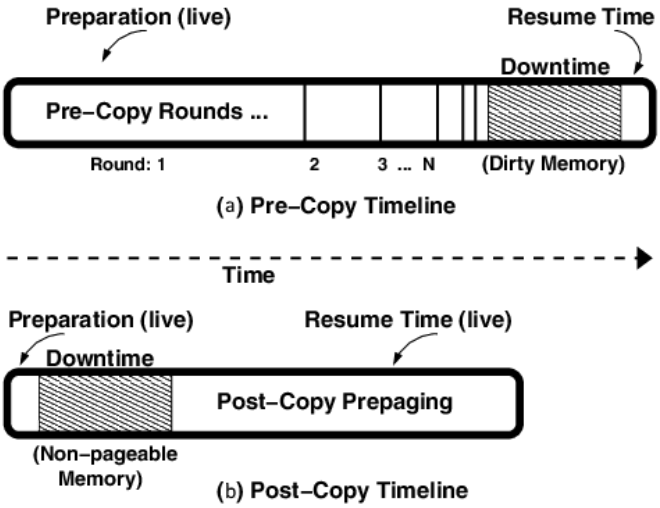


Figure 2.4: Pre-Copy vs Post-Copy Timeline.

2.4 Resource Allocation

In cloud computing, resource allocation is the process of allocating available physical resources to the needed virtual machines and applications. Hypervisors that we mentioned before provide mechanisms to map virtual machines to physical resources. Even though the mapping process is generally hidden from the end user, this process is crucial to the cloud providers for efficiency.

Resource demands of VMs may change with user requests all the time. However, since the available resources on a physical machine (PM) is limited, the mapping of VMs needs to be changed dynamically to provide the requested resources. VM live migration technology enables cloud providers to change the mapping between VMs and the physical machines while applications are running [20, 21].

Resource allocation process requires lots of different policies to decide how to change mapping adaptively so that the resource demands of the VM are met while the number of active physical machines are minimized for energy efficiency. The diverse set of applications running on VMs and their changing workloads cause VMs to demand heterogeneous resources. Also, different physical machines create heterogeneous environment in datacenters. This heterogeneity makes the mapping process even a harder problem for the cloud providers.

For resource allocation strategies in the literature, there are two main goals tried to be achieved: overload avoidance and green computing. In order to eliminate overload, the requested resources of VMs should not exceed the available resources of the host. Otherwise, the host is overloaded and this may cause SLA violations. For green computing, the number of active servers needs to be minimized. If the host is underutilized and there are other available hosts, which are not fully utilized, then the VMs on underutilized host can be migrated to these other hosts so that the unnecessary hosts can be put into sleep mode.

There is a trade off between overload avoidance and green computing. In order to avoid overloaded PMs, we need to keep utilization of PMs quite low so that in case of an unexpected resource demand from a VM, the requested resource can be provided to the VM without causing any overload. However, for green computing, we need to keep utilization of PMs as high as possible to make efficient use of their energy.

Most of the frameworks proposed for efficient resource allocation are based on four different steps, which are: Detection of overloaded hosts, detection

of underloaded hosts, detection of VMs which should be migrated from overloaded/underloaded hosts, and detection of hosts which should be selected as the destination host. Different frameworks are proposing different algorithms for each of these steps. In this thesis we focus on first two steps of the resource allocation which are overload and underload host detection.

Since live migration of a VM is triggered with overloaded or underloaded host detection, the correct prediction of the hosts' CPU loads is one of the most crucial step in resource allocation.

2.5 Summary

As we discussed in this chapter, we see that virtual machine allocation is very important for cloud providers for efficient use of their resources while providing better quality of service. Virtualization technology and live migration technique enable providers to allocate virtual machines dynamically as the workload changes. We also see that overloaded and underloaded host detection for efficient VM allocation is a well researched topic in the literature. As a new approach, we propose an improved predictive host detection algorithm for energy efficient VM allocation. In the following chapters, we explain our proposed solution in detail and show that it is a feasible solution by providing simulation results.

Chapter 3

Proposed Solution

As we described in the previous chapter, resource allocation for virtual machines has been studied quite substantially by academia and is also an important problem that needs to be handled by cloud providers. Since resource allocation is an NP hard problem, there are lots of different frameworks proposed for the solution. Some of the proposed algorithms in the literature are trying to detect the overloaded and underloaded hosts by predicting the future CPU utilization of the hosts [3, 4, 12]. Even though prediction based overloaded host detection algorithms perform better than the rest, they also have some deficiencies which can be improved by using statistics.

In this chapter, we first describe the basic methods that we used in our algorithm LRAPS. Then we discuss why current CPU prediction algorithms have some deficiencies and how our algorithm solves these problems. We also describe the implementation of our prediction algorithm as part of dynamic VM resource allocation and consolidation process. We finish the chapter with a summary.

3.1 Local Regression

Local regression is originated as LOWESS (LOcally WEigted Scatter plot Smoother) and in the literature it is also called as Loess. After Cleveland [22] proposed the Loess procedure in 1998, it is mostly used as a scatter-plot smoother. Since Loess can work with multivariate data, it is also used as a modeling tool by statisticians. Local regression mainly tries to fit simple models to localized subsets of the data to get a function which approximates the original data. For simplicity we can consider a fixed model of the form

$$Y_i = f(x_i) + \varepsilon_i \tag{3.1}$$

In this form Y_i are observations of a response, the $f(x_i)$ is the unknown function and ε_i represents the random errors in the observation not included in the x_i.

We assume that errors are independent and identically distributed, which means that each random variable has the same probability distribution as the other and all are mutually independent, with mean 0 and variance $var(\varepsilon_i) = \sigma^2$.

In local regression there is no need to specify global assumptions about the function f, instead we assume that f can be well approximated locally with a simple parametric function. This is why our CPU prediction algorithm can work better with local regression, because the workload can be chaotic which means that it is not always possible to find a global function, instead local values represent the future workload better. To understand local regression better, we need to explain Taylor's theorem.

Taylor's theorem [23] says that any continuous function can be approximated with a polynomial. Assume f is a real function on $[a, b]$, $f^{(K-1)}$ is continuous on $[a, b]$, $f^{(K)}_{(t)}$ is bounded where $t \in (a, b)$, for any points $x_0 < x_1$ in $[a, b]$ there is an x point between $x_0 < x < x_1$ so that

$$f(x_1) = f(x_o) + \sum_{k=1}^{K-1} \frac{f^{(k)}(x_0)}{k!}(x_1 - x_0)^k + \frac{f^{(K)}(x)}{K!}(x_1 - x_0)^K \tag{3.2}$$

if $\sum_{k=1}^{K-1} \frac{f^{(k)}(x_0)}{k!}(x_1 - x_0)^k$ is the function of x_1, Equation 3.2 can be written as follows:

$$\mathcal{P}_{K+1} = \{f(x) = a_0 + a_1 x + ... + a_K x^K, (a_0, ..., a_K)' \in \mathbb{R}^{K+1}\} \tag{3.3}$$

Next we define the method to obtain a Loess smooth for a target covariate x_0.

The first thing to do in local regression is to specify a weight function. For Loess only local values within the smoothing window should be used. The smoothing window can be expressed as $[x_0 + h(x_0), x_0 - h(x_0)]$ where $h(x_0)$ is span value. Since we just want to consider the values in the smoothing window, we need to define our weight function $f(x_0)$ accordingly.

The weight function can be found by considering that the function should be 0 outside of $[-1, 1]$. For instance, Tukey's tri-weight function is

$$W(u) = \begin{cases} (1 - |u|^3)^3, & |u| \leq 1 \\ 0, & |u| > 1 \end{cases}$$

Then the weight sequence can be found by

$$w_i(x_0) = W\left(\frac{x_i - x_0}{h(x)}\right) \tag{3.4}$$

Window can be defined similar to k nearest points. That is, $a \times 100\%$ of data needs to be included in local regression.

In local nearest points, $f(x)$ can be approximated with a polynomial. A quadratic approximation, like in Equation 3.5, can be used

$$f(x) \approx \beta_0 + \beta_1(x - x_0) + \frac{1}{2}\beta_2(x - x_0)^2 \text{ for } x \in [x_0 - h(x_0), x_0 + h(x_0)] \quad (3.5)$$

In order to get the loess estimate $\hat{f}(x_0)$ we need to find $\beta = (\beta_0, \beta_1, \beta_2)'$ that minimizes the expression below

$$\hat{\beta} = argmin_{\beta \in \mathbb{R}^3} \sum_{i=1}^{n} w_i(x_0)[Y_i - \{\beta_0 + \beta_1(x - x_0) + \frac{1}{2}\beta_2(x - x_0)\}]^2 \quad (3.6)$$

And we define $\hat{f}(x_0) = \hat{\beta}_0$.

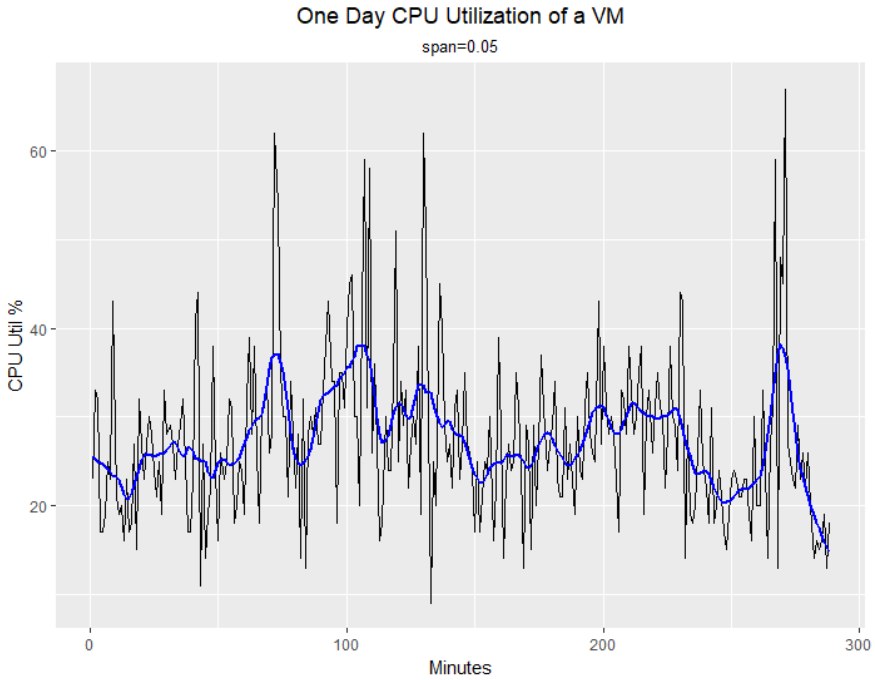
In local regression there are two arguments need to be decided. These are smoothing parameter (span) and the degree of the Loess polynomial.

In a Loess fit, the smoothing parameter a tell us the proportion of the observations that will be used in local regression. Since a is the proportion of the data, it is specified between 0 and 1. If $a = 1$, that means all of the observations will be used in the local regression.

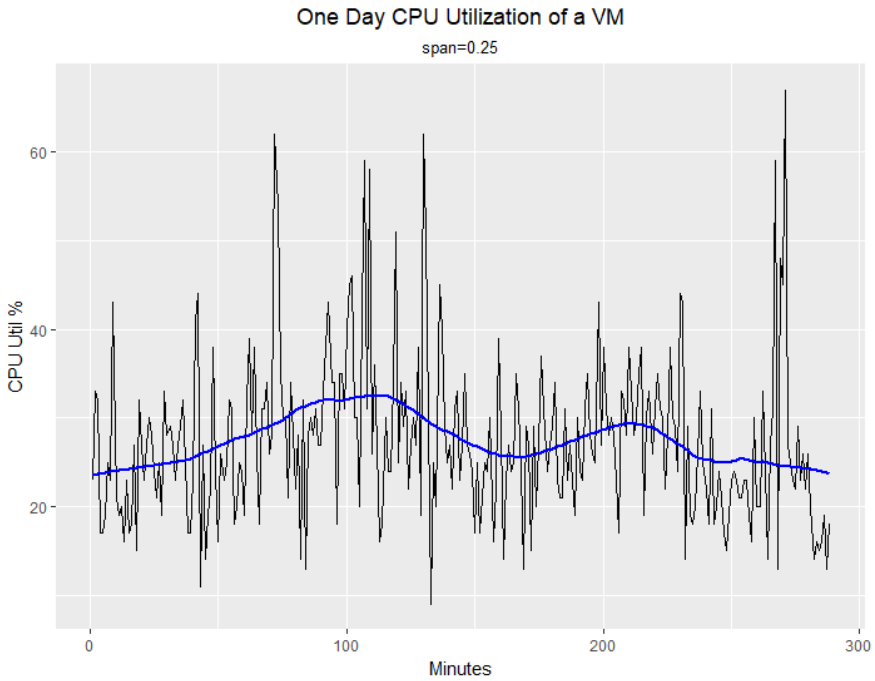
Defining the optimum span is very important for the accuracy of the local regression. As a increases the estimate becomes smoother.

In [3], authors are utilizing local regression to fit a trend polynomial to the last k observations of the CPU utilization for every dataset. However, span value affects the prediction results significantly and therefore using the same span value for every dataset decreases the prediction accuracy.

In the figures below, we see Loess smooths for one day of CPU utilization of a VM from PlanetLab dataset [24]. The smooths are created using span values of 0.05, 0.25, 0.75 and 0.95. Different polynomial degrees are also plotted. Black line represents the actual workload of the VM on that day, whereas blue line is the predicted workload through local regression with the span value on it.

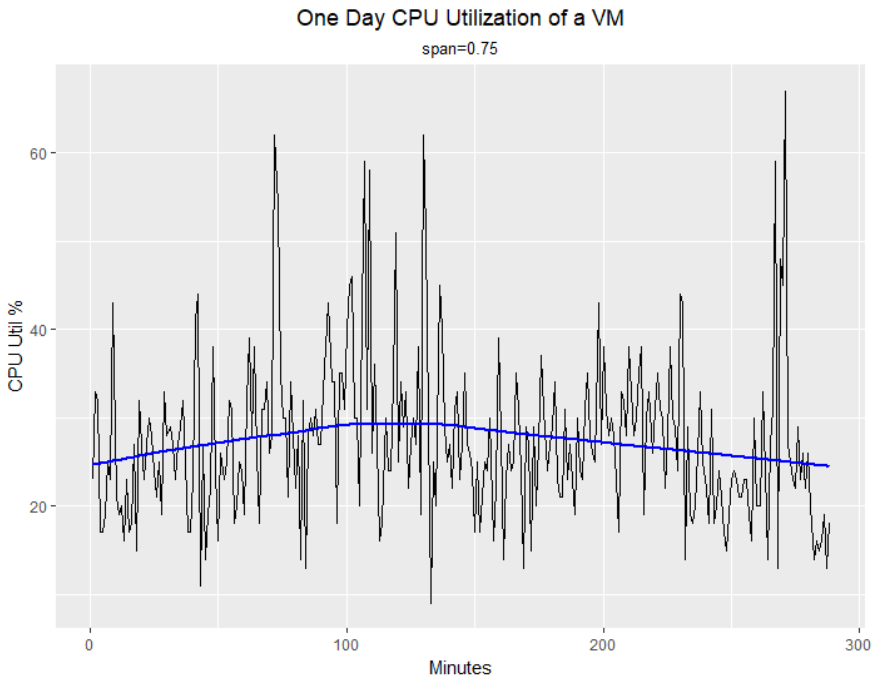


(a) Degree=1 , span=0.05.

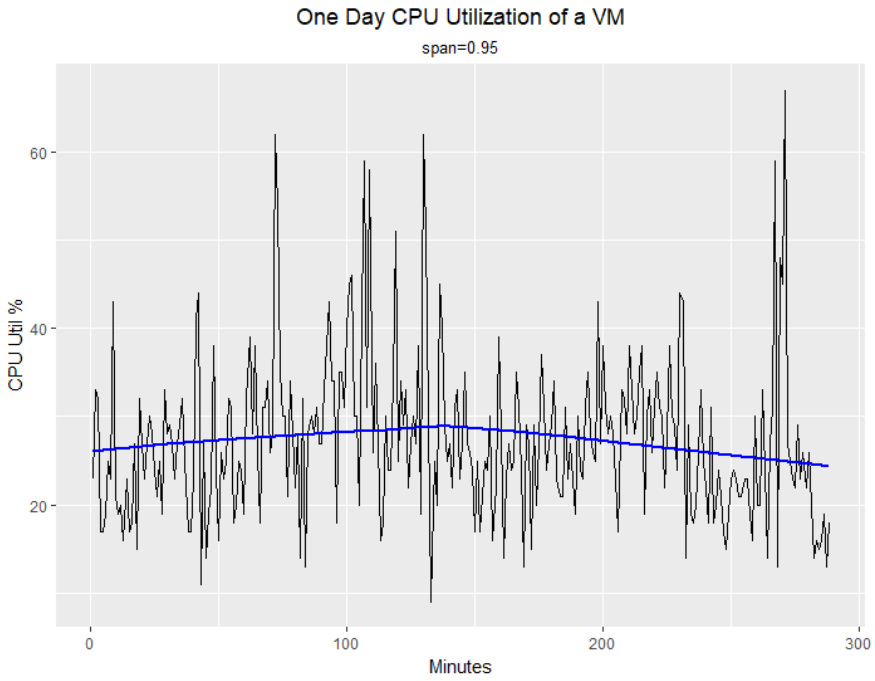


(b) Degree=1 , span=0.25.

Figure 3.1: Loess Prediction where h=0.05 and 0.25, d=1.

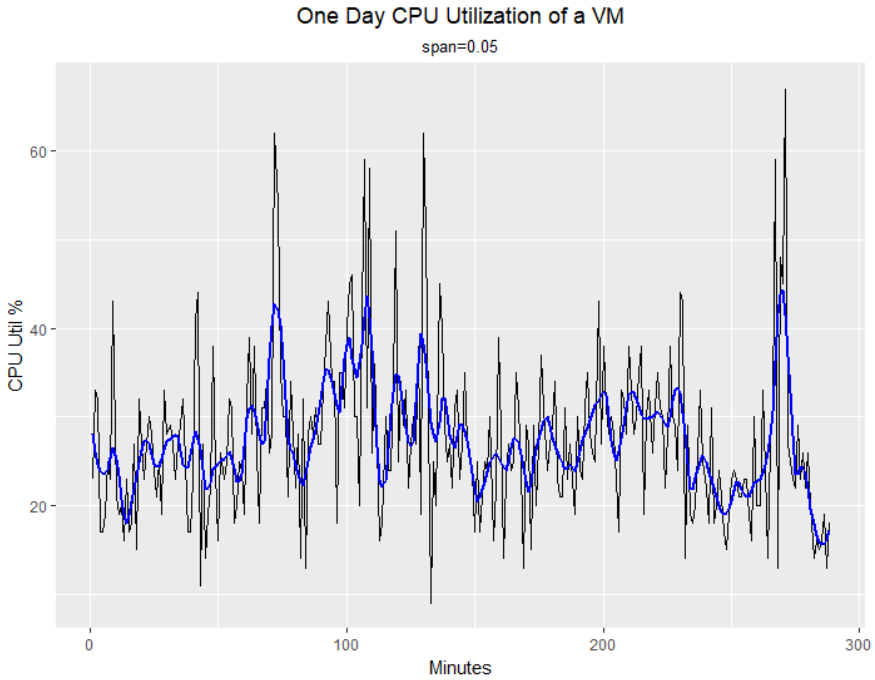


(a) Degree=1 , span=0.75.

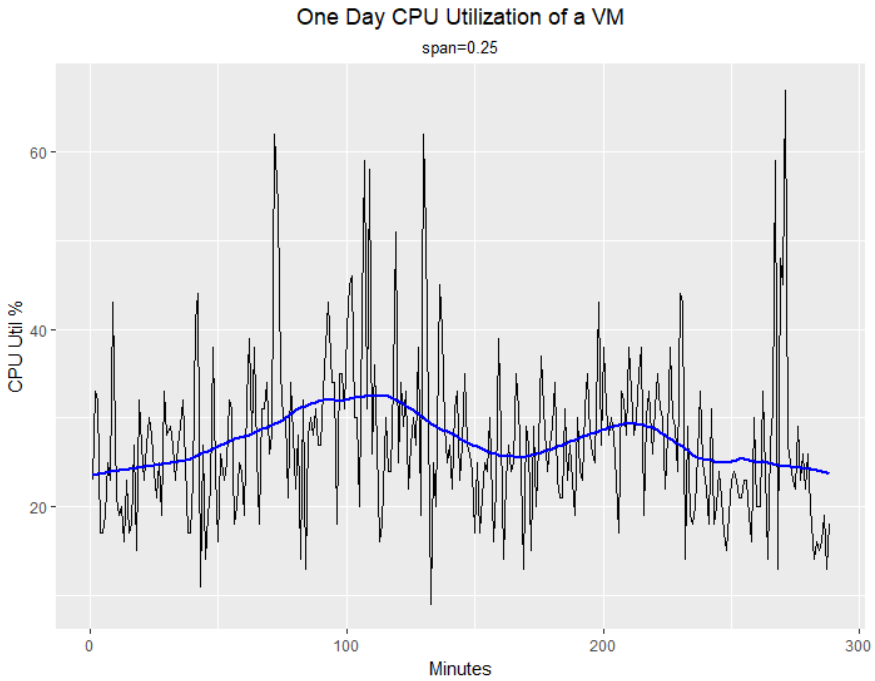


(b) Degree=1 , span=0.95.

Figure 3.2: Loess Prediction where h=0.75 and 0.95,d=1.

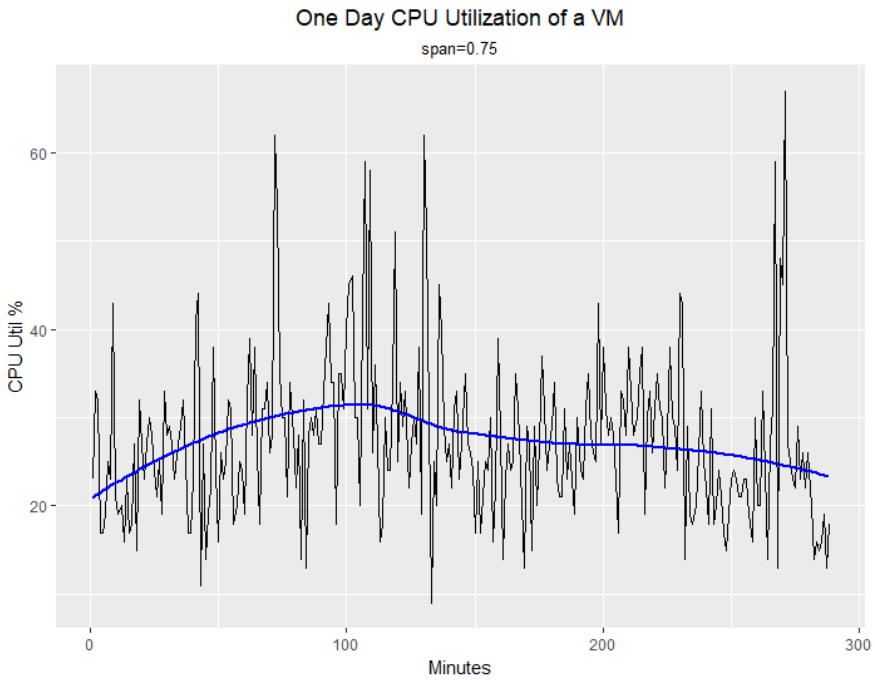


(a) Degree=2 , span=0.05.

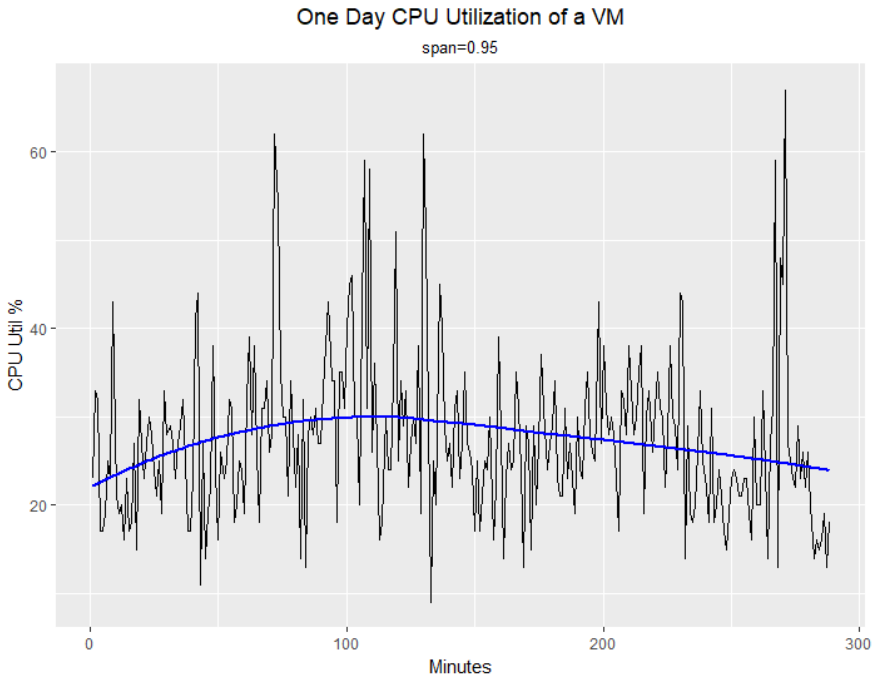


(b) Degree=2 , span=0.25.

Figure 3.3: Loess Prediction where h=0.05 and 0.25,d=2.



(a) Degree=2 , span=0.75.



(b) Degree=2 , span=0.95.

Figure 3.4: Loess Prediction where h=0.75 and 0.95,d=2.

As seen in the figures, if we choose a small span value, there will not be enough data near x_0 for the accurate fit. Using less data near x_0 causes over fitting of local regression and large variance. If we choose a large span value, there will be too much data near x_0 to be considered. Since local regression cannot give response to the latest values, it loses information and becomes over smoothed.

As we mentioned before there are two parameters that affect the trade-off between variance and bias in Loess. Besides the span value, the degree of the polynomial is the another parameter that has an impact on local regression. Choosing polynomial degree of 1 distorts the peaks in the interior of the configuration of observations, which results in low bias. On the other hand choosing polynomial degree 2 removes the distortion, but results in higher biases at the boundaries.

In general it is not always possible to find the optimum span value and the degree of polynomial automatically without plotting and testing different values. However, for the workload of PMs in a datacenter, setting a span value with eye that gives better prediction is not possible, since the environment is too dynamic and there is a large number of physical machines used in datacenters.

In order to improve the local regression based prediction algorithm, we decided to use dynamic span value which changes depending on the dataset. With dynamic span value, we can predict the next CPU usage more accurately and detect overloaded hosts more effectively. In the next chapter we discuss different methods for automated span selection.

3.2 Span Selection

We mentioned the importance of the span value (smoothing parameter) for local regression. In this part we will discuss different selection methodologies for smoothing parameter.

Assume the related regression model as

$$Y_i = f(x_i) + \varepsilon_i \tag{3.7}$$

where $f(x)$ is an unknown function of interest and the errors are IID. A smoothing spline estimate $\hat{f}_\lambda$ for f is defined as the minimizer of the penalized criterion:

$$\frac{1}{n}\sum_{i=1}^{n}\{y_i - f(x_i)\}^2 + \lambda \int_a^b \{f''(x)\}^2 dx \tag{3.8}$$

In the Equation 3.8 above, λ is the span value (smoothing parameter) which is a positive constant. As we mentioned in the previous section λ controls the trade-off between the bias and the variance of $\hat{f}_\lambda$. Therefore selecting an appropriate value of λ is an important problem.

In the literature mainly six smoothing parameter selection methodologies are studied and compared for regression based estimation. Three of the selection methodologies are called as classical whereas the rest are so-called second generation. The three classical methods are cross-validation (CV), generalized cross-validation (GCV) and Mallows' C_p criterion. The other three second-generation methods are Akaike information criterion (AIC), risk estimation using classical pilots (RECP), and exact double smoothing (EDS).

Even though the second generation methods are proposed to solve the drawbacks of classical methods like high variability and under smooth tendency, they have received considerably less attention for smoothing splines.

First we define some notations to understand the methods clearly. Let $y = (y_1, ..., y_n)^T$, $f = (f(x_1), ..., f(x_n))^T$ and $\hat{f}_\lambda = (\hat{f}_\lambda(x_1), ..., \hat{f}_\lambda(x_n))^T$. Let S_λ be the hat matrix that maps y into $\hat{f}_\lambda = S_\lambda y$. And $S_\lambda = (I + \lambda K)^{-1}$, where I is the identity matrix and K is a matrix depending only on $x_1, ..., x_2$.

Since we focus on GCV (generalized cross validation) method in our thesis, we will give some information about cross-validation and generalized cross-validation.

Cross-Validation: In cross validation, points $\{x_i, y_i\}_{i=1}^n$ are separated one by one to find a λ which minimizes the residual sum of squares. The cross validation performance can be expressed as

$$CV(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left\{y_i - \hat{f}_\lambda^{(-i)}(x_i)\right\}^2 \equiv CV(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{y_i - \hat{f}_\lambda(x_i)}{1-(S_\lambda)_{ii}}\right\}^2 \tag{3.9}$$

In Equation 3.9, $\hat{f}_\lambda$ is the fit for measurements $\{x_i, y_i\}_{i=1}^n$ with smoothing parameter λ. $\hat{f}_\lambda^{(-i)}$ is the fit calculated by separating the ith data point, and $(S_\lambda)_{ii}$ is the ith diagonal element of smoother matrix S_λ.

Generalized Cross-Validation: GCV is a modified version of cross-validation method mentioned above. The score of GCV can be obtained by replacing the denominators of CV which is $1-(S_\lambda)_{ii}$ with their average value $1 - n^{-1}tr(S_\lambda)$. Hence, the general cross validation performance score can be expressed as

$$GCV(\lambda) = \frac{1}{n}\frac{\sum_{i=1}^{n}\left\{y_i - \hat{f}_\lambda(x_1)\right\}^2}{\{1-n^{-1}tr(S_\lambda)\}^2} = \frac{n^{-1}||(I-S_\lambda)y||^2}{[n^{-1}tr(I-S_\lambda)]^2} \tag{3.10}$$

In [25], extensive experiments were conducted for these different smoothing parameter selection methodologies. MSE values of the selection methodologies are used to understand the quality of the estimation $\hat{f}$. Also, in order to understand the significance of the MSE value differences between methods, a paired Wilcoxon tests were applied.

The scores in Table 3.1 were obtained according to means of the averaged Wilcoxon test results.

According to same study [25], especially for small samples, GCV is recommended as being the best selection method. By considering the extensive study done for the different selection methodologies and the dataset that we will face with, we decided to use GCV as our automated span selection method.

Table 3.1: Means of the Averaged Wilcoxon Test Results for Smoothing Parameter Selection Methods.

Method	Noise level	Spatial variation	Variance function	Overall
CV	5.125	6.083	5.583	5.597
GCV	2.792*	2.625*	3.000*	2.806*
C_p	5.667	6.083	5.875	5.875
AICc	5.667	5.958	5.875	5.833
RECP	5.667	6.083	5.875	5.875

3.3 Proposed Algorithm: LRAPS

Predicting future CPU utilization of a host is very critical for improving energy efficiency of datacenters and for maintaining quality of service levels. If the future CPU utilization of a host is known, VMs located on that host can be migrated to less loaded machines in advance before SLA violations start occurring, or a machine can be avoided as a target for migration.

Our proposed algorithm LRAPS (Local Regression Automated Parameter Selection) is based on local regression technique to predict future CPU utilization. It uses historical CPU usage of a host to make a proper prediction for future CPU usage. In this section, we first explain how our algorithm works. Then we explain the differences between regular local regression and our algorithm LRAPS. Then we explain how our algorithm is used as part of live migration process.

As discussed in the previous section, there are parameters of local regression method that need to be decided before the prediction is done, such as polynomial degree and span. Unlike polynomial degree, span value may have a dramatic effect on prediction results. For example, if the CPU utilization history has a lot of short spikes and we are trying to find the general trend in data to make sound decisions, choosing a small span value may result in overfitting. Additionally, choosing a small span value may generate wrong and frequent overload condition alarms as part of VM migration process, which would cause unnecessary migrations.

Since span value can significantly affect prediction performance, we decided

to use an algorithm which will automatically find the optimum span value for local regression. With the addition of automated span selection, we ended up with our LRAPS algorithm. LRAPS algorithm fits a local regression to utilization measurements and automates the parameter selection process, so that during the CPU prediction of a host only the necessary CPU usage measurement data is used. The extensive comparison study [25] conducted on several different selection methods revealed that for both large and small data samples, GCV is recommended as being the best span selection method.

LRAPS algorithm has two parts: local regression based prediction and automated span selection with GCV. For Loess prediction, we can assume that there will be historical data provided to the function, which includes past CPU utilization data of the hosts. The percentage of historical data that will be used in Loess prediction is found by $a \times 100$, where a is the result of GCV based span selection method. GCV based method returns the span value which has the least square error. This part is the most crucial part of our algorithm since it differs from the regular local regression algorithm used in the literature and it significantly affects the performance. In Figure 3.5, we can see the difference between the Loess prediction with static span value and dynamic span value.

The Figure 3.5 shows the one day CPU utilization of a VM in a datacenter. Black lines represent the actual workload, whereas blue and red lines are the predicted utilization values. Blue line represents the local regression based prediction with degree $d = 2$ and span value $h = 0.25$. This static span value is the one used in local regression methods in the literature [3]. Red line is the one that we calculated with our LRAPS prediction algorithm. With our algorithm optimum span value is found as $h = 0.543$ for the current dataset. Hence instead of using a static span value, we used a dynamic span value found with GCV.

In [3], if the future CPU utilization of a host is higher than a threshold such as shown in Figure 3.5, the host is labeled as overloaded and some of the VMs running on it are migrated. However, using a static span value for all different datasets may cause wrong predictions. Even though the blue line in Figure 3.5 seems to cross the threshold, actually the overall trend of a host (red line) does

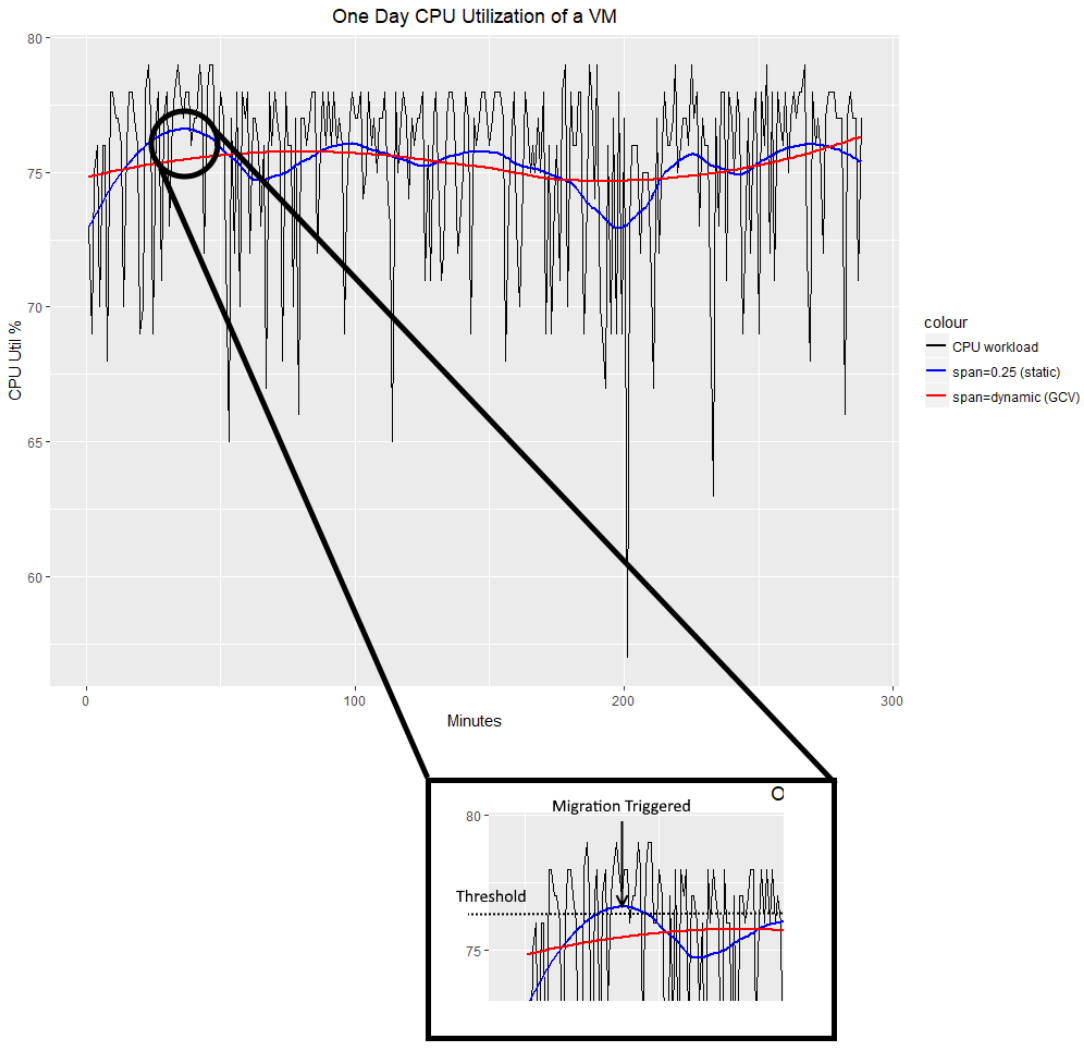


Figure 3.5: Local Regression with static span vs dynamic span value.

not indicate any overloaded situation. So using a static span value may trigger unnecessary VM migrations which result in excessive power consumption.

CPU utilization of a server, which accumulates through time, may change dramatically for different time intervals and therefore needs to be reanalyzed for accurate prediction. Hence, different span values need to be used for each server and also for different time periods in a single server.

As we discussed earlier, dynamic VM allocation consists of mainly four different steps: Overload/underload detection and labeling, VM selection from labeled hosts and destination host selection. In pseudo-codes that we will explain, the methods $VMSelection$ and $VMAllocation$ represent last two steps of the allocation process. Since these are not our main concern in this thesis, we only show which algorithms that we used.

Algorithm 1: VM live migration

```
1 for each host in hostList do
2     if hostOverloaded(host) OR hostUnderloaded(host) then
3         VM = VMSelection(host);
4         otherHosts = hostList − host;
5         h = VMAllocation(otherHosts, VM);
6         migrate(VM, h);
7     else
8     end
9 end
```

Algorithm 2: Overloading detection function

Input : a host object from datacenter *host*
Output: boolean

```
1  if host.utilizationHistory < 10 then
2  |   if host.currentUtilization > 0.7 then
3  |   |   return true;
4  |   else
5  |   |   return false;
6  |   end
7  else
8  |   predictedUtil = LRAPS(host);
9  |   if predictedUtil * safetyParameter ≥ 1 then
10 |   |   return true;
11 |   else
12 |   |   return false;
13 |   end
14 end
```

Algorithm 3: LRAPS Local Regression Automated Parameter Selection

Input : a host object
Output: predicted utilization

```
1  span = GCV(host.utilizationHistory);
2  newUtil = host.utilHistory(1 : span * length);
3  estimates = getLoessParameterss(newUtil);
4  predictedUtil =
     estimates[0] + estimates[1] * (length + migrationIntervals);
5  return predictedUtil;
```

Our algorithm is part of the live migration process. Every host in the datacenter is being checked if any overload or underload has occurred. At the beginning, since there is not enough data for utilization prediction, we use a static threshold for the upper limit for the normal-load of a host, which is 0.7. If the current CPU utilization of a host is higher than 0.7, then the host is labeled as overloaded. When enough CPU utilization history gets accumulated, our prediction algorithm starts to run. Before doing predictions, we first try to decide which span value should be used for Loess function. In order to decide the span value, all utilization history is passed through the generalized cross validation method. The result of the GCV function tells us the span value. By using this span value we trim the CPU utilization history. Then, predictions can be started. The trimmed history values are sent to Loess function which predicts the next CPU utilization. After the next CPU utilization is predicted, we multiply it with a safety parameter which is defined previously system-wide. If the result of multiplication is higher than or equal to 1, then the host is labeled as overloaded and the rest of the migration process continues.

Besides the host load condition detection policy that we propose, VM selection policy needs to be decided as well as part of VM consolidation process. During the consolidation process, after our algorithm has detected an overloaded host, VM selection policy needs to start running to select a VM from the overloaded host to migrate to another host. For VM selection policy, we decided to use Minimum Migration Time (MMT). MMT policy tries to find the VM that requires the minimum time to complete its migration to the destination host among other VMs. MMT satisfies the condition in Equation 3.11 to find a VM.

$$v \in V_j | \forall a \in V_j, \frac{RAM_u(v)}{NET_j} \leq \frac{RAM_u(a)}{NET_j}, \tag{3.11}$$

where V_j represents VMs allocated to the host j, $RAM_u(a)$ is the RAM utilized by VM a and NET_j is the available spare network bandwidth for host j.

For all the experiments which we will explain in the next chapter, we used MMT as our VM selection policy for fair comparison.

3.4 Summary

In this section, we have first discussed the local regression based prediction method. We showed that different span values cause very different prediction results. Then we showed various span selection methods for local regression. We analyzed local regression prediction results with static span value and the dynamic span value. Then we gave details of our algorithm LRAPS and its incorporation into VM allocation process with pseudo-codes.

Chapter 4

Simulation Experiments and Evaluation

We used CloudSim toolkit [26] as the simulator to simulate and evaluate our method. CloudSim is gaining popularity in cloud computing community thanks to its support for flexible, scalable, efficient and repeatable evaluation of provisioning policies for different applications. With this toolkit anyone can implement his own energy saving and SLA policies to compare with other algorithms. We implemented our LRAPS algorithm in CloudSim and we also extended some of the built-in provisioning algorithms of CloudSim to compare with our algorithm.

In this chapter we first give information about the simulation environment, datasets,and metrics we used for our experiments, and then we provide and discuss the results of our extensive simulation experiments.

4.1 Simulation Environment

4.1.1 Simulation Architecture

In Figure 4.1 you can see the layered architecture of CloudSim. There are three main layers included in the toolkit which are: User code, CloudSim layer and CloudSim core simulation engine layer. CloudSim simulation engine is the core of the toolkit which provides basic simulation capabilities to the above layers. This layer can be modified by the developers but this is not recommended since the other layers provide enough flexibility for any kind of simulation experiments. CloudSim layer is the most important layer which handles a lot of fundamental tasks performed in cloud datacenters such as VM provisioning, resource allocation, cloudlet execution, VM management, event handling. Any developer can extend this layer for the custom simulation experiment settings. In this thesis we also extended some of its built in functionalities to implement and test our LRAPS algorithm.

At the top layer, user code, there are basic entities which can be modified easily by any user. These entities can be listed as number of machines, their CPU and power consumption specifications, number of tasks, virtual machines and their requirements, number of users and some scheduling policies. After we implemented our algorithm in CloudSim layer, we changed the entities in user code layer to compare and see the performance of the algorithm for different environments.

There are several different datacenter modeling types implemented in CloudSim. Since we want to compare the energy performance of the VM allocation algorithms, we decided to use power aware datacenter in simulation experiments.

There are two main policies in VM provisioning service: VM allocation policy and VM selection policy. VM allocation policy mainly responsible for the first and the second steps of the dynamic VM allocation process which are deciding

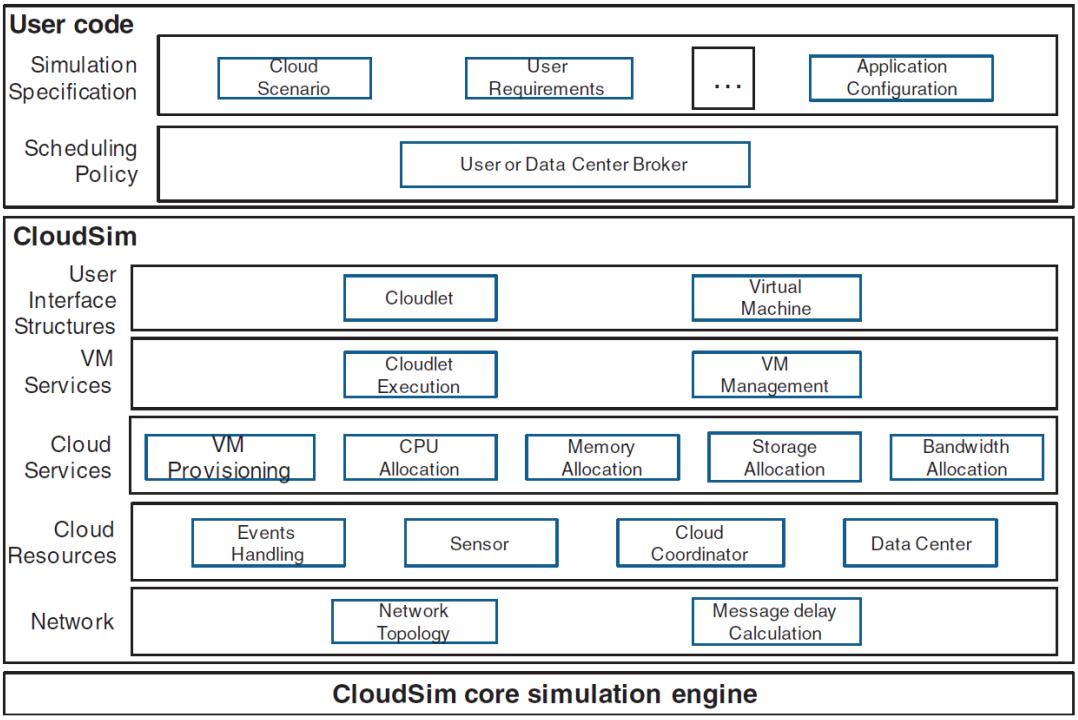


Figure 4.1: Live CloudSim Layered Architecture.

overloaded and underloaded hosts. Since our algorithm tries to improve the decision process, we extended the VM allocation policy. Some VM allocation policies are already implemented in CloudSim environment. However, policies used by large public providers are not publicly available, and therefore implementation of these algorithms are not provided in CloudSim. Besides VM allocation, VM selection is responsible for the third step of the dynamic allocation process which is determining the VMs which will be migrated from overloaded or underloaded host. Since we do not focus on VM selection policy, we decided keep it fixed as MMT algorithm, mentioned in the previous chapter, for all the experiments.

CloudSim is implemented in Java and the overall class design diagram is shown in the Figure 4.2. After we give information about the important components of CloudSim design, we explain which parts we extended for the experiments.

Components:

- **Cloudlet:** This class models the application services.

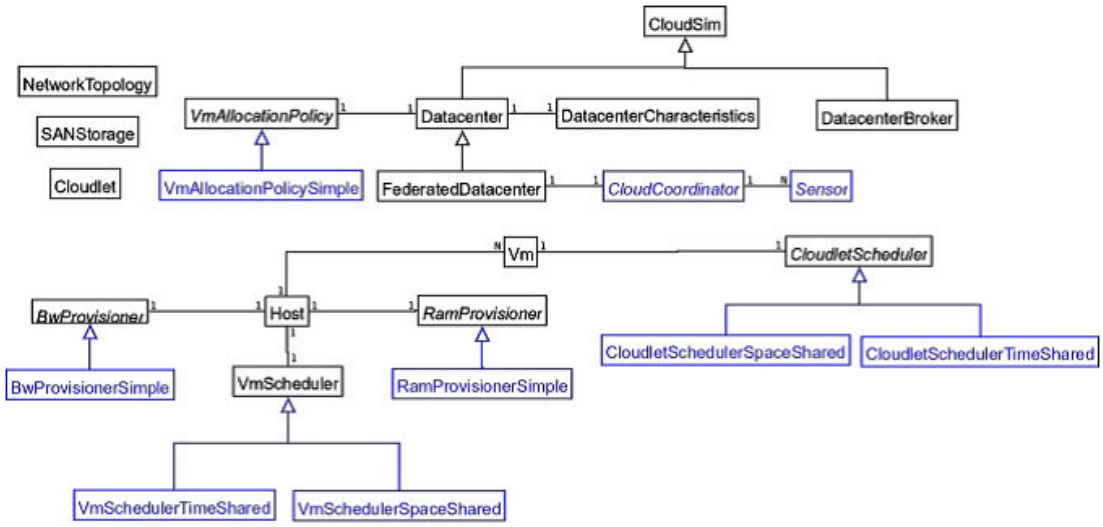


Figure 4.2: CloudSim Class Diagram.

- **CloudletScheduler:** This is an abstract class that is extended by different policies. And also responsible for the sharing of processing power between Cloudlets in a VM.
- **Datacenter:** This represents the core hardware level services that are offered by providers.
- **Datacenter or CloudBroker:** This class maintains the agreements between SaaS and Cloud providers.
- **DatacenterCharacteristics:** This class contains data center resource configurations.
- **Host:** This class models a physical storage or a server.
- **NetworkTopolgy:** This class includes information of network behavior.
- **RamProvisioner:** This class is responsible for RAM allocation to the VMs.
- **VM:** This class models a VM.
- **VMAllocationPolicy:** This class is responsible for the allocation of VMs to the hosts.

- **VMScheduler:** This is responsible for allocating processor cores to VMs.

For our experiments we changed the VM, Datacenter and Cloudlet characteristics in the user code layer. Then we implemented our LRAPS algorithm in the CloudSim layer by adding new classes to VMAllocationPolicy.

4.1.2 Simulation Metrics

In our experiments we used three metrics to evaluate the performance of the algorithms.

Total Energy Consumption: This metric measures the quantity of energy consumed by the data center within the simulation time. In order to make our simulation similar to real life, we used two different server models. Half of the servers are HP Proliant ML110 G4 servers and the other half are HP Poliant ML 110 G5 servers. Both servers have 1 Gb/s network bandwidth. In CloudSim [26], the CPU frequencies of the servers are mapped onto MIPS ratings. Each of two cores of HP Poliant ML110 G5 servers have 2260 MIPS, each of the two cores of HP Proliant ML110 G4 servers have 1860 MIPS. The power models used in [3] are applied to our simulation experiments. Energy consumption of the servers according to the utilization rates is shown in Table 4.1.

Table 4.1: Power Consumption by the Selected Servers at Different Load Levels in Watts.

Server	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP ProLiant G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G5	93.7	97	101	105	110	116	121	125	129	133	135

Total Number of Migrations: Live virtual machine migration has an adverse impact on the performance of applications located in virtual machines. Voorsluys et al. [27] conducted several experiments to understand the degree of impact. As a result of the study they have found that performance degradation and downtime vary too much with the application's behavior. However, applications with variable workloads, like web-applications, the average performance

degradation with downtime is approximated as 10% of the CPU utilization, which means that during the migration 10% of the CPU capacity is used by the migration process. In our simulation this amount of CPU capacity is allocated to a VM on the destination host during migration. This enables us to simulate the environment more realistically and observe the migration effect on the performance. Since virtual machine migration causes extra downtime in the CPU utilization of VM, it may also cause some SLA violations. In order to prevent this, we need to minimize the number of VM migrations.

SLA Violation Rate: Quality of Service (QoS) is very crucial goal for both the cloud providers and the users. Cloud providers have to maintain a level of quality of service for their users. In cloud computing QoS requirements can be expressed as minimum throughput or maximum response time in the SLA form. Since these two parameters can significantly vary depending on the application, it is necessary to use more generic and independent metrics to determine the SLAs violation for any VM. In this thesis we defined and used two such metrics. These metrics are: (1) the percentage of time, when the utilization of active hosts are 100%, called SLA violation Time per Active Host (SLATAH); and (2) the total performance degradation of VMs caused by live migrations, called Performance Degradation due to Migrations (PDM).

$$SLATAH = \frac{1}{N}\sum_{i=1}^{N}\frac{T_{s_i}}{T_{a_i}} \ , \ PDM = \frac{1}{M}\sum_{j=1}^{M}\frac{C_{d_j}}{C_{r_j}} \tag{4.1}$$

Above, N is the number of hosts, T_{s_i} is the total time during which host i has experienced the utilization 100%, leading to an SLA violation; T_{a_i} is the active time of host i; M is the number of VMs; C_{d_j} is the estimated performance degredation of the VM j caused by migrations; C_{r_j} is the total CPU capacity requested by the VM j during its lifetime. At the end we propose a combined metric that includes both $SLATAH$ and PDM. We call the combined metric as SLA Violation (SLAV), which is calculated as below:

$$SLAV = SLATAH \times PDM \tag{4.2}$$

In our experiments we use this metric to achieve a more general comparison.

4.1.3 Experimental Setup

We simulated a datacenter consisting of 300 heterogeneous hosts: half of them are HP Proliant ML110 G4 servers with 1860 MIPS each core and the other half are HP Poliant ML110 G5 servers with 2660 MIPS each core. Each server have 2 cores, 4GB memory and 1 Gb/s network bandwidth. Power consumption characteristics are stated in Table 4.1.

We used four different VM instances corresponding to Amazon EC2 [1]: High CPU medium instances (2500MIPS, 0.87 GB); Extra Large Instance (2000 MIPS, 1.74GB); Small Instance (1000 MIPS, 1.74GB), and Micro Instance (500 MIPS, 613 MB). At the beginning, VMs are allocated according to the resource requirements defined for them. For every experiment we used a total of 500 VMs.

Since we try to compare our prediction algorithm with others as fairly as possible, we kept other provisioning algorithms the same for all the experiments. As VM selection policy we used Minimum Migration Time for all benchmark algorithms.

We evaluated our proposed algorithm LRAPS over a time span of 24 hours for different workloads.

4.1.4 Datasets

Since datasets directly affect the results of prediction algorithms and allocation process, we used three different real world publicly available datasets and one randomly generated dataset.

PlanetLab Workload: PlanetLab trace [24] was collected during ten days of March and April 2011 from 11,746 VMs. Most of the machines where the data is collected are hosted by research institutions. CPU utilization of VMs are measured every five minutes throughout one day. Hence, for each VM there are 288 data points which corresponds to one day. For our experiment we used the datasets of 9 April, 11 April, 12 April and 20 April.

Random Workload: Random workload is generated for 800 VMs. Memory and CPU utilization are included in the dataset, however we only used CPU utilization. CPU utilization of each VM follows a uniform distribution.

Materna Datacenter: Two of the three available Materna datasets [28] are used in our simulation. The first trace consists of 520 VMs whereas the second trace consists of 527 VMs. Traces were collected from the distributed Materna Datacenters in Dortmund over a timespan of three months. The traces were taken on a VMWare ESX environment where the physical resources are 49 hosts, 69 CPU cores and 6780 GB RAM. Each dataset contains CPU, Memory, Disk and Network usage information of the virtual machines.

Bitbrains: The dataset contains the performance metrics of 1,750 VMs of a distributed datacenter from Bitbrains [29], which is a service provider in managed hosting and business computation for enterprises. Performance metrics are recorded during a month of 2016. Like Materna dataset, the dataset contains CPU, Memory, Disk and Network usage information.

4.2 Simulation Experiment Results

Using the datasets described in Section 4.1.4, we have simulated the five different host overload detection algorithms: THR, LR, MAD, IQR and LRAPS. In the following figures and tables we compare our algorithm LRAPS with LR, THR 0.8, THR 0.7, MAD and IQR methods in the given simulation environment.

Table 4.2: Energy Consumption (kWh).

Dataset	LR	LRAPS	MAD	IQR	THR 0.7	THR 0.8
9-Apr	66.84	58.57	76.01	77.49	82.58	77.8
11-Apr	69.95	58.1	82.43	83.36	90.27	83.1
12-Apr	75.16	63.06	87.3	88.99	94.73	87.53
20-Apr	61.36	47.75	72.19	73.79	77.49	72.13
Bitbrains	52.5	52.2	54.61	59.05	59.99	58.13
Random	35.37	30.68	45.97	48.86	45.58	41.81
Materna-1	43.76	43.61	51.78	52.41	54.62	50.6
Materna-2	44.74	46.48	52.82	54.74	56.63	54.46

First, we have conducted simulation experiments for measuring energy consumption (Figure 4.3, Table 4.2). As expected, static threshold (THR) method generally consumes too much power. Since the threshold is static, even a small spike in the CPU utilization triggers VM migration. THR 0.7 algorithm triggers the migrations too early so that none of the physical servers in datacenter can reach to utilization higher than 0.7. Since the maximum utilization of the servers stuck below 0.7, more sleeping servers are activated to provide the requested resource demand which resulted in higher power consumption. THR 0.8 also gave bad results, however, better than THR 0.7, since the utilization of the servers can go higher. Adaptive threshold algorithms MAD and IQR showed better results than THR, since they are adapting the threshold according to the workload. Both LR and LRAPS algorithms outperformed the other algorithms by minimizing the power consumption for all datasets. The results show that predicting future CPU utilization provides better overload detection than the others. Most importantly the results also show that better overload prediction can be achieved by utilizing automated smoothing parameter selection methods like GCV. Our LRAPS algorithm reduces the total power consumption of a datacenter by about 23%, 28%, 25%, and 22%, compared with THR 0.8, THR 0.7, IQR, and MAD, respectively. Our LRAPS algorithm also outperforms the closest competitor LR by reducing the power consumption about 10%.

Then we compared the algorithms according to SLA violations. SLA violation is another important metric and there is a tradeoff between energy consumption

Table 4.3: SLA Violation.

Dataset	LR	LRAPS	MAD	IQR	THR 0.7	THR 0.8
9-Apr	9.81%	9.31%	10.39%	9.94%	10.08%	10.17%
11-Apr	9.60%	8.53%	10.09%	10.21%	10.09%	9.98%
12-Apr	9.57%	9.21%	10.15%	10.04%	10.00%	10.24%
20-Apr	10.27%	9.13%	10.25%	10.36%	10.62%	10.19%
Bitbrains	10.86%	11.09%	10.79%	10.54%	10.08%	10.69%
Random	12.89%	11.25%	10.69%	10.32%	11.12%	12.81%
Materna-1	10.12%	9.96%	10.30%	10.37%	10.29%	10.19%
Materna-2	10.16%	9.68%	10.23%	10.27%	10.05%	10.39%

and SLA violation. While reducing power consumption, cloud providers want to keep the quality of service as high as possible as well. In Figure 4.4 and Table 4.3, we can see the overall SLA violation metric values for different algorithms. SLA violation of the algorithms are very similar to each other. All of them are almost placed around 10% except for random workload. Although most of the algorithms performed similar in all datasets, one thing obvious is that LRAPS algorithm decreased SLA violation slightly better than the rest. This is because our algorithm can predict the trend in CPU utilization more accurately and can avoid overloaded machines better. Decreasing overloaded machines results in fewer SLA violations. LRAPS reduces SLA violations about 7.6%, 5.1%, 4.8%, 5.9% and 6% compared with THR 0.8, THR 0.7, IQR, MAD and LR, respectively.

We also measured the number of migrations for all the algorithms. As seen in Figure 4.5 and Table 4.4, LRAPS algorithm drastically decreases the total number of virtual machine migrations. Again this result shows how effective our prediction algorithm is. LRAPS reduces the number of VM migrations about 31.7%, 36.2%, 32.4%, 32.2% and 27%, compared with THR 0.8, THR 0.7, IQR, MAD and LR, respectively.

Table 4.4: Number of VM Migrations.

Date	LR	LRAPS	MAD	IQR	THR 0.7	THR 0.8
9-Apr	11261	8318	11564	11510	12723	11678
11-Apr	12130	6910	12461	12551	13617	12345
12-Apr	12842	8889	13162	13172	14184	12945
20-Apr	11065	5717	11172	11513	12124	11447
Bitbrains	12162	11362	12143	11978	12327	11873
Random	2872	744	5303	5535	5310	4839
Materna-1	9917	9524	10632	10298	11153	10283
Materna-2	9569	10552	10708	11068	11441	10920

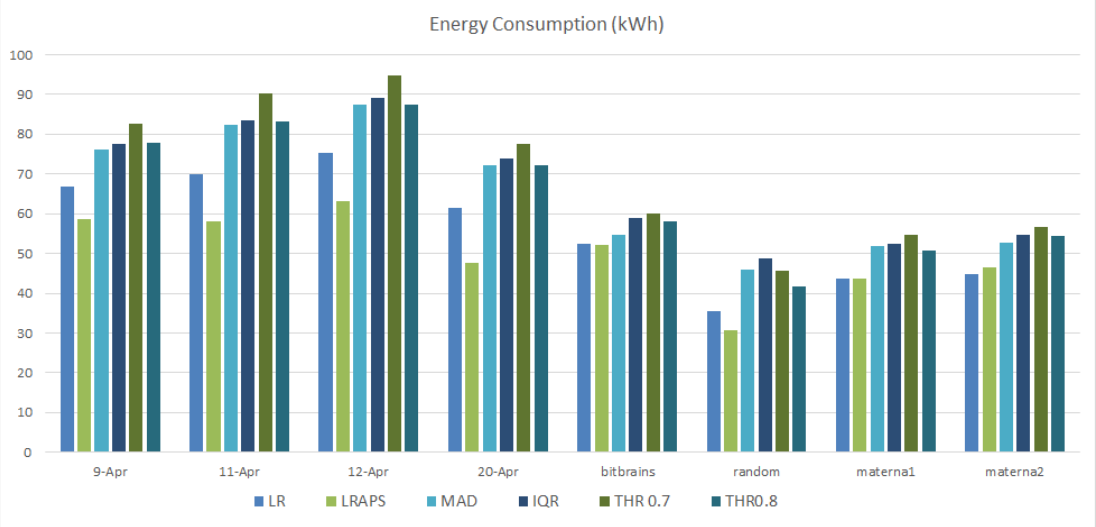


Figure 4.3: Energy Consumption.

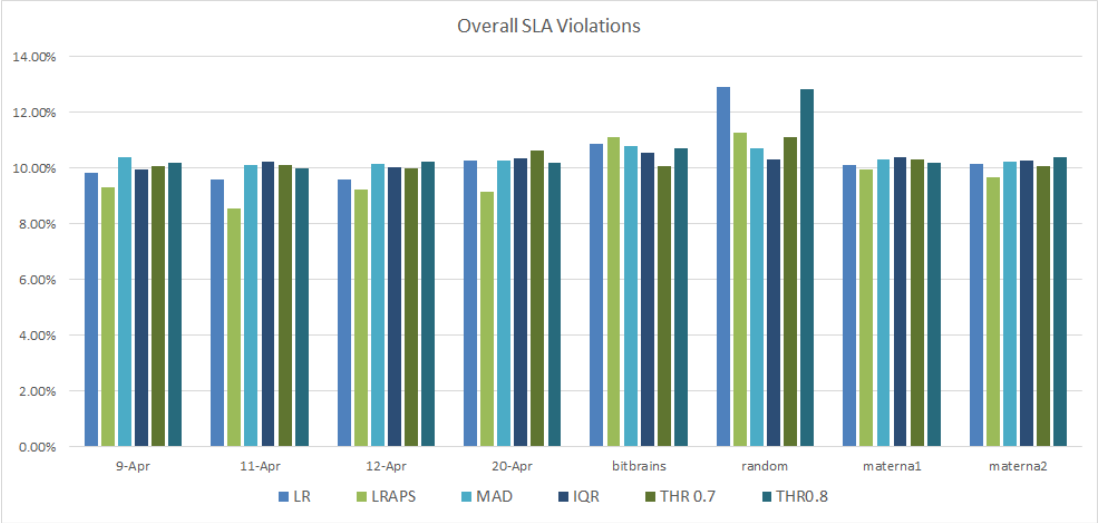


Figure 4.4: Overall SLA Violations.

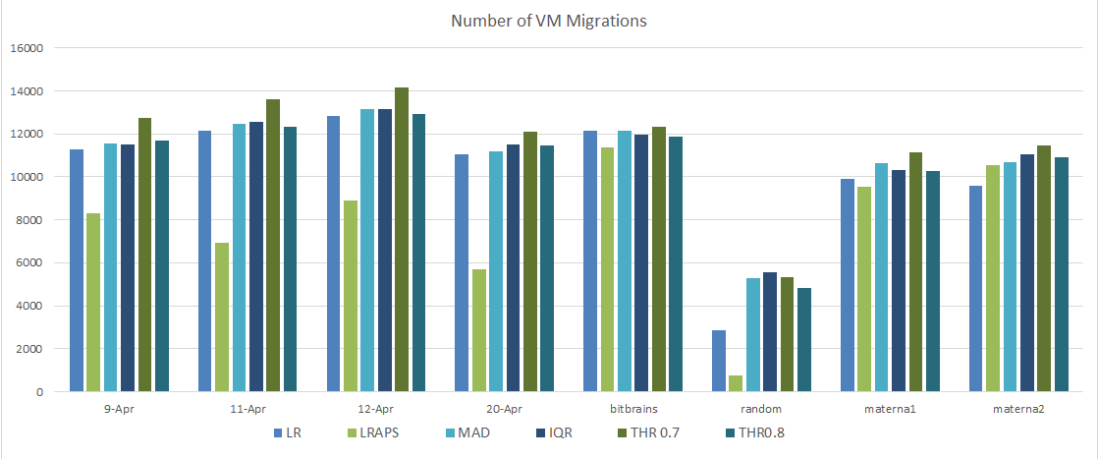


Figure 4.5: Number of VM Migrations.

4.3 Summary

[h] In this section we first gave detailed information about CloudSim simulation environment. We extended some of the core functionality of the simulation environment. We described the layered structure of the CloudSim toolkit and how it can be modified by developers to customize and extend functionality. For our experiments we decided use three metrics that are similar to the metrics used in the literature. For realistic comparison and performance analysis, we used three different real world traces and one synthetic workload. The results showed that our LRAPS algorithm decreases power consumption and SLA violations significantly compared to LR algorithm and other algorithms, thanks to our dynamic span selection method.

Chapter 5

CONCLUSION

In this thesis we showed that allocating VMs to the minimum number of physical server is very important for cloud providers in terms of energy consumption and SLA violation. We analyzed current allocation policies, especially overload detection algorithms in the literature. Even though local regression (LR) based CPU prediction algorithm performs better than other algorithms, we discovered that LR algorithm has some deficiencies like static span value. To solve this problem, we proposed our prediction algorithm LRAPS (Local Regression Automated Parameter Selection).

Our prediction algorithm is based on local regression, however instead of using a static span value, we used generalized cross validation (GCV) method to find dynamic span value. We discovered that using dynamic span value makes CPU prediction better such that sound migration decisions can be made. And that decisions may provide better utilization. After we construct our LRAPS algorithm, we implemented it into CloudSim simulation environment. To implement our algorithm we extended some of core classes of CloudSim toolkit.

For realistic experiment results, we used three different real world workload traces and one synthetic workload. We simulated one day of a datacenter with 500 VMs and 800 hosts. With automated span selection method, our local regression

based prediction algorithm (LRAPS) out performed other algorithms for all three different metrics; energy consumption, SLA violation and number of migrations.

5.1 Future Work

As future work, the dimension of our algorithm can be increased. Different resource types of VM like memory, I/O and network can be considered for the same allocation strategy. Other policies of consolidation process can be analyzed like VM selection policy and destination host selection policy. Simulation environment can be modified such that long term simulations can be performed like 30 days. This might help us to analyze the experimental results in detail.

Besides VM allocation, a new technology called container allocation is emerged recently. Similar experiments can be done for this new technology, since the allocation processes are similar to each other.

Bibliography

[1] "Amazon ec2." http://aws.amazon.com/ec2/. Accessed: August 4, 2017.

[2] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," *10th IEEE/ACM International Conference, Cloud and Grid Computing (CCGrid)*, pp. 577 – 578, 2010.

[3] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *IEEE Transactions on Medical Imaging*, vol. 24, pp. 1397 – 1420, 2012.

[4] F. Farahnakian, P. Liljeberg, and J. Plosila, "Lircup: Linear regression based cpu usage prediction algorithm for live migration of virtual machines in data centers," *Software Engineering and Advanced Applications (SEAA)*, pp. 357 – 364, 2013.

[5] K. Maurya and R. Sinha, "Energy conscious dynamic provisioning of virtual machines using adaptive migration thresholds in cloud data center," *International Journal of Computer Science Mobile Computing*, vol. 3, pp. 74 – 82, 2013.

[6] A. Beloglazov and R. Buyya, "Openstack neat: A framework for dynamic and energy efficient consolidation of virtual machines in openstack clouds," *Concurrency and Computation: Practice and Experience*, vol. 27, pp. 1310 – 1333, 2015.

[7] S. S. Masoumzadeh and H. Hlavacs, "An intelligent and adaptive threshold-based schema for energy and performance efficient dynamic vm consolidation," *Energy Efficiency in Large Scale Distributed Systems Springer*, pp. 85 – 97, 2013.

[8] L. Salimian, F. S. Esfahani, and M. Nadimi-Shahraki, "An adaptive fuzzy threshold-based approach for energy and performance efficient consolidation of virtual machines," *Computing*, vol. 98, pp. 641 – 660, 2015.

[9] A. Horri, M. S. Mozafari, and G. Dastghaibyfard, "Novel resource allocation algorithms to performance and energy efficiency in cloud computing," *The Journal of Supercomputing*, vol. 69, pp. 1445 – 1461, 2014.

[10] C. Hsu, K. D. Slagter, S. Chen, and Y. Chung, "Optimizing energy consumption with task consolidation in clouds," *Information Sciences*, vol. 258, pp. 452–462, 2014.

[11] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Transactions of Parallel and Distributed Systems*, vol. 24, pp. 1107–1117, 2013.

[12] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, pp. 155–162, 2012.

[13] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy efficient cloud resource management," *IEEE INFOCOM*, pp. 386–391, 2014.

[14] F. Farahnakian, T. Pahikkala, P. Liljeberg, and J. Plosila, "Energy aware consolidation algorithm based on k-nearest neighbor regression for cloud data center," *2013 IEEE/ACM 6th International Conference on Utility And Cloud Computing*, pp. 256–259, 2013.

[15] Z. Zhang, H. Wang, L. Xiao, and L. Ruan, "A statistical based resource allocation scheme in cloud," *International Conference on Cloud and Service Computing*, pp. 266–273, 2011.

[16] "Vmware - virtual machine." https://www.vmware.com/. Accessed: August 23, 2017.

[17] "Hyper-v." https://www.microsoft.com/tr-tr/cloud-platform/server-virtualization/. Accessed: August 17, 2017.

[18] "Vmware player." https://www.vmware.com/products/workstation-player.html. Accessed: August 17, 2017.

[19] "Parallel desktops." https://www.parallels.com/eu/products/desktop/. Accessed: June 5, 2017.

[20] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Network System Design and Implementation*, 2005.

[21] M. Nelson, B. H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *USENIX Annual Technical Conference, General Track*, 2005.

[22] W. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *Journal of the American statistical association*, vol. 368, pp. 829 – 836, 1979.

[23] R. Irizarry, "Lecture notes in applied nonparametric and modern statistics." http://rafalab.github.io/pages/754/ Accessed: August 5, 2017.

[24] K. Park and V. S. Pai, "Comon: A mostly-scalable monitoring system for planetlab," *Operating Systems Review*, vol. 40, pp. 65–74, 2006.

[25] D. Aydın, M. Memmedli, and R. Omay, "Smoothing parameter selection for nonparametric regression using smoothing spline," *European journal of pure and applied mathematics*, vol. 6, 2013.

[26] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Journal of Software: Practice and Experience*, vol. 41, pp. 23–50, 2011.

[27] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation.," in *CloudCom*, 2009.

[28] "Materna dataset." http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna. Accessed: October 6, 2017.

[29] "Bitbrains dataset." http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains. Accessed: October 20, 2017.