Carnegie Mellon University

CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Master of Science

TITLE **VPLACEMENT: Contention Aware**

Virtual Machine Placement System

PRESENTED BY Ankur Kumar Sharma

ACCEPTED BY THE DEPARTMENT OF

Information Networking Institute

[illegible]	5/6/2014
THESIS ADVISOR	DATE
Rajeev Gandhi	5/6/2014
ACADEMIC ADVISOR	DATE
[illegible]	5/7/14
DEPARTMENT HEAD	DATE

APPROVED BY THE COLLEGE COUNCIL

Vijayakumar Bhagavatula	May 8, 2014
DEAN	DATE

VPLACEMENT: Contention Aware Virtual Machine Placement System

Submitted in partial fulfillment of the requirements for

the degree of

Master of Science

in

Information Networking

Ankur Kumar Sharma

B.Tech., Computer Science, National Institute of Technology, Allahabad
M.S., Information Networking, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

May, 2014

Acknowledgements

Working on Master's thesis was an excellent experience for me. Working on new ideas and analyzing new problems is something that i have always cherished. I would like to thank everyone who helped me in this wonderful journey of sharing and executing new ideas.

I would like to express deep gratitude for my advisor Professor Hyong Kim. His support and guidance helped me a lot during the project. I was especially impressed by his practical approach. He taught me many valuable lessons about work ethics and practical engineering research. He helped me a lot during the thesis and was always available for guidance. I would also like to thank my reader Professor Rajeev Gandhi for his support and valuable feedback on the thesis.

I would like to thank my lab mates in theone network research group for all the help and support that they provided. I would like to thank John Payne for being so helpful in data center setup that helped me run my experiments. He was always available for clarifying my doubts and helped me a lot in understanding the vtactic system which is the basis of my work. I would also like to thank senbo fu for helping with the vtactic setup. I would also like to thank the INI staff for being so helpful during the whole process.

I would like to thank my parents and my brother for supporting me over the years for my education.

The thesis was not funded.

Abstract

Maximizing the number of cohosted virtual machines (VMs) while still maintaining the desired performance level is a critical goal in cloud. As we pack more virtual machines on a physical machine (PM), the resource contention increases, thereby affecting the response time.

This virtual machine placement problem has been vastly studied and most of effort has been in either allocating more resources to virtual machines (resizing) or migrating them to a higher capacity PM based on the resource demand estimation. Studies have also shown that in the presence of resource contention the resource demand estimation mechanisms could predict more resource requirement than actually needed. Hence deciding virtual machine placement and allocated resources based on utilization estimation could lead to inefficient usage of PM resources.

We propose a novel approach to solve this problem which focuses on overall application response time rather than individual virtual machines. Large scale applications are deployed as multi-tier components. These components interact with each other so that application can perform its task. Our placement algorithm uses the dependency relationship between these components to understand application response time behavior. Our solution focuses on reducing the performance degradation because of resource contention.

We propose a VM placement system termed as Vplacement. This system uses the traffic analysis to understand the dependency relationship between application components. This dependency relationship and traffic analysis provides some vital data like impact of component processing time on application response time, the probability of resource contention between a pair of component nodes (coArrival Probability) etc. The impact and coarrival probability is used by the placement engine of Vplacement to minimize the degradation of application performance because of resource contention by cohosting the low impact component nodes together.

Table of Contents

List of Tables

List of Figures

1 Introduction

Virtualization has become important component of data centers. Live migration [10] of virtual machines helps improve the manageability of data centers. The concept of hosting multiple VMs on same PM is especially very important. It plays a critical role in efficient utilization of hardware resources. But this sharing of physical resources amongst virtual machines leads to a degradation in performance because of resource contention. This can impact the overall application response time.

1.1 Multi-tier Application

Large scale applications are deployed as multi-tier components. These components interact with each other so that application can perform its task. For example, figure 1.1 shows a typical multi-tier application deployment for a web application. Where presentation layer serves the static content, application layer generates dynamic content and data tier interacts with the database. In a virtualized data center these components will be deployed as VMs.

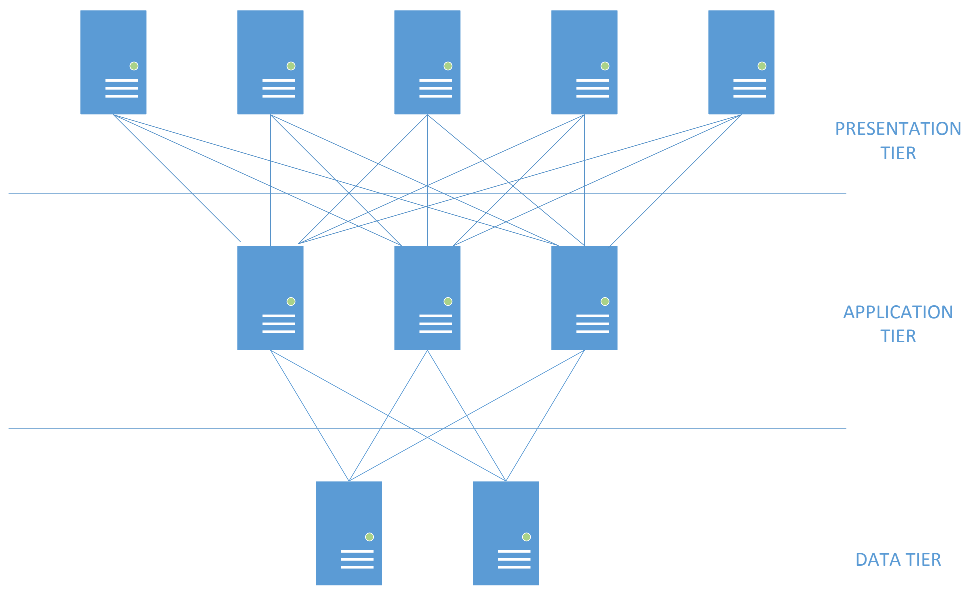


Figure 1.1: Multi-tier Application.

1.2 Virtual Machine Placement

The decision to place a particular VM to a particular host is known as VM placement. Key challenge here is to maximize the number of cohosted VMs (consolidation ratio) while still maintaining the required performance level. It is seen as a multi-dimensional bin-packing problem [6]. The resource requests of VMs are considered as d-dimensional vectors with non-negative entries. The resource available at each PM is considered as a d-dimensional vector [6].

Figure 1.2 illustrates a simple VM placement scenario [7]. We have three physical machines (HostA, HostB and HostC) that are used to run five virtual machines

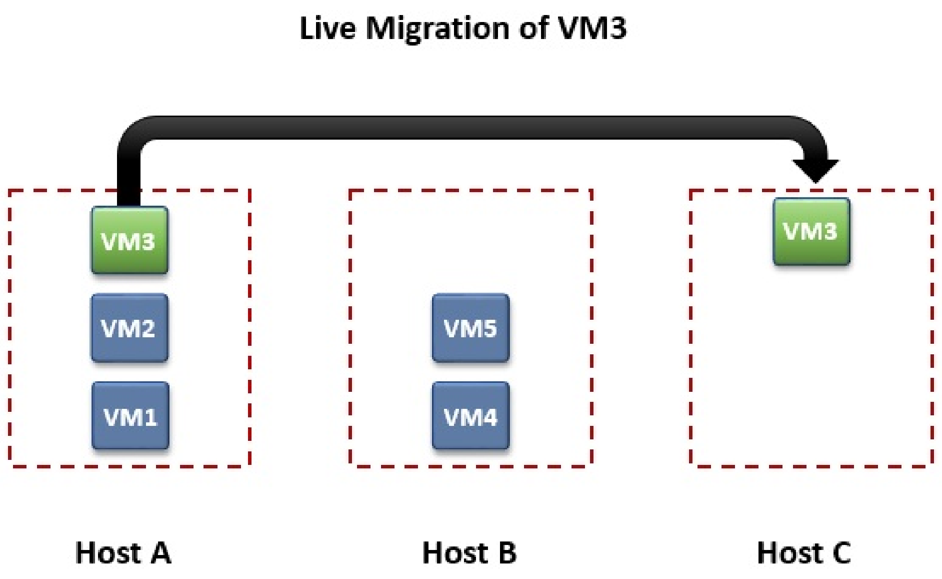


Figure 1.2: VM Placement example [7].

(VM1, VM2, VM3, Vm4 and VM5). Initial placement is to place VM1, VM2, VM3 on HostA and VM4, VM5 on Host B. In order to improve the performance the VM placement algorithm may trigger the migration of VM3 to Host C.

1.3 Resource Contention

Virtualization technology provides strong isolation amongst the VMs. For example Security isolation prevents one virtual machine from accessing data of another. Fault isolation prevents failure in one machine to impact the other [14].

Modern virtualization technologies do not provide effective performance isolation. While the hypervisor slices the system resources and allocates them to virtual machines. But still the execution of one virtual machine can significantly impact the performance of other because of shared use of resources of the system [24]. For example, two virtual machines running on different cores (on same host) may experience a significant reduction in performance because of increase in miss rate of last level cache access [19]. Similarly, consider two VMs each of which requires 100 milliseconds to process a request. The response time could become 200 seconds if both of them run on same CPU and are assigned cpu cycles in equal share [22].

The resource contention may also lead to incorrect estimate of resource requirement. Many VM placement algorithms rely on estimating the resource requirement of virtual machines, because of presence of other virtual machines this estimation may indicate a much higher resource requirement then what is actually needed [12].

1.4 Traffic Analysis

Traffic Analysis plays an important in our placement system. We use Vtactic application performance management system [21]. This system analysis the inter VM traffic to derive the dependency amongst various component nodes of an application. Vtactic uses the probability of coarrival to understand the impact of resource contention on the VM. It derives the various dependencies between application components to estimate the overall application response time. Vtactic uses its analysis further to derive the impact of a component node on total application response time. The impact, coarrival probability and application components are critical parameters for the placement algorithm used by VPlacement System.

The impact, coarrival probabilities and application components are taken as input by Vplacement's placement algorithm. The component nodes are prioritized based on their impact and then with the help of coarrival probabilities it places the VMs so that impact of resource contention on application can be minimized.

1.5 Problem Statement

In this thesis, we build a VM placement system termed as Vplacement that focuses on optimizing the application response time while still trying to achieve a high consolidation ratio. Given the current available PMs and initial resource requirement of VMs, system tries to find the optimal VM-PM mapping which reduces the impact of resource contention on application response time.

1.6 Contributions

1.6.1 VM placement system

We developed a VM placement system which integrates traffic analysis component (Vtactic), Placement Engine Component (Vplace) and a Cloud Management System (Openstack). This system interacts with vtactic and runs the placement algorithm, once the VM-PM mappings are decided then it interacts with openstack to physically migrate the VMs.

1.6.2 libopenstack

We developed a library in C which provides a simple get/put interface to interact with Openstack Controller. Lack of any such library in C Programming Language motivated us to go ahead with the development.

1.6.3 Placement Engine

We propose a new placement algorithm which reduces the effect of resource contention on application response time. This Placement Engine uses the derived component impact and coarrival probabilities to reduce the contention for high impact VMs.

1.7 Thesis Organization

The document is organized as follows. In Chapter 2 we study the background work done in the domain virtual machine placement. In Chapter 3 we introduce the Vplacement system. In Chapter 4 we explain the basic concepts in Vtactic system.

In Chapter 5 introduce openstack and provide details on our work on instance launch. In Chapter 6 we give an overvie wof libopenstack library. In Chapter 7 we explain the functioning of Vplace module. In Chapter 8 we exlain the placement algorithm used by Vplacement system. In Chapter 9 we discuss the experimental results. In Chapter 10 we conclude the thesis.

2 Background

The domain of VM placement has been vastly studied. The problem has been approached from different perspectives. Some algorithms focus on improving the VM response time by either moving to a different host or by allocating it more resources. Some algorithms focus on reducing the network traffic by cohosting those virtual machines which has higher inter VM traffic. While some algorithms focus on the energy consumption and try to colocate the VMs so that more servers can be switched off to save power.

Besides the placement algorithms there has been studies on the impact of resource contention on VM performance and evaluating the current VM placement algorithms.

We can roughly classify the related work into following categories: Resource Contention, Optimizing VM response Time, Optimizing Network Traffic, Optimizing Energy Consumption and Evaluation.

2.1 Resource Contention

Researchers have done several studies to estimate the impact of resource contention on application response time. Koh et al.[14] studies the effects of performance interference by looking at system-level workload characteristics. In this study authors characterized the workloads that generate intense performance interference. Authors also developed a performance prediction mechanism. Mars et al.[16] introduces Bubble-Up a characterization methodology that predicts the performance degradation because of contention in memory subsystem. This system basically works in 2 steps. Firstly, it evaluates the pressure on the memory subsystem an application generates. Secondly, it measures how much an application suffers because of pressure on memory subsystem. Isci et al.[12] focuses on evaluation of cpu utilization when VMs are colocated. It demonstrates that measured CPU utilization may provide a poor estimate of actual requirement. Authors introduce a new mechanism that keeps track of hypervisor scheduling metric to estimate the actual CPU requirement of a VM. Corradi et al.[11] evaluates the current VM placement algorithm. It introduces the challenges which consolidation and performance requirement incurs. It studies the current vm placement algorithms and concludes that they most of them cover only a few dimensions of resource usage and some algorithms do no consider the effect of resource contention. Using openstack as cloud management software they deploy virtual machines and study the impact of cohosted VMs on performance degradation. Sukwong et al.[22] researchers in this paper introduce a new system termed as SageShift. The paper introduces an admission control module termed as Sage and a SLA aware hypervisor scheduler termed as Shift. Sage calculates the coarrival probability VM requests to evaluate its impact on the target SLA. In shift the researchers changed the KVM hypervisor scheduler to make it SLA aware so that VM which is closer to missing its deadline can get more CPU share.

2.2 Optimizing Vm Response Time

This study can be further put into two categories. First category focuses on studying the resource usage patterns of the virtual machines and then resizing or migrating the VMs to improve performance. Second category uses the impact of intereference to figure out correct vm-host mapping.

2.2.1 Resource Usage Based

There has been several efforts to optimize VM response time based on the resource usage pattern. Wood et al.[26] proposes a system termed as sandpiper which monitors the resource usage of each VM, it monitors cpu,memory and network bandwidth usage. It detects if the resource demand exceeds a certain threshold. It mitigates the problem by moving VMs to a higher capacity host. Bobroff et al.[7] focuses on forecasting the resource requirement based on current usage. It recommends sorting of VMs by resource requirement and then running FFD/BFD heuristics. Tang et al.[23] works at the application level. It works in multiple, where in each round it estimates the maximum application resource demand which current placement can fulfill. If the current placement cannot fulfill the application resource requirement then it triggers a VM migration and tries to stop unproductive instances. Calcavecchia et al.[8] Introduces a backward speculative placement (BSP) algorithm which projects past demand behavior of a VM to a candidate host. This algorithm focuses on satisfying CPU demand. Bellur et al.[6] approaches this problem as a multi-dimensional bin packing problem. The resource request of VMs are treated as d-dimensional vectors with non-negative entries. The resource available at each PM (bin) is considered to be a d-dimensional vector. The goal here is to minimize the number of bins such that for every bin the sum of vectors placed in that bin is coordinate-wise no greater than the bin's vector. Chandra et al.[9] focuses on capturing dynamic workload of web applications. It proposes a mechanism to dynamically allocating resources to VMs based on application workload. Padala et al.[20] also focuses on dynamically changing the resource allocation for dynamic workload changes. It creates a model for dynamic relationship between the application's resource allocations and its performance under current workload. It further predicts the required resources needed by application to meet its performance requirement.

2.2.2 Interference Based

Some researchers have introduced the usage of vm interference as a metric for dynamic resource allocation. Nathuji et al.[19] introduces a new framework termed as Q-Cloud. It is a Qos aware control framework that tunes resource allocation to mitigate performance interference effects. It tries to allocate underutilized resource to achieve the Qos target.

2.3 Optimizing Network Traffic

Another criteria which researchers have used for VM placement is to optimize the network bandwidth. Meng et al.[17] colocates the VMs which interact with each other more. It localizes large chunk of traffic so that load on high level network switches is reduced. Wang et al[25] works in similar direction. It triggers Vm consolidation based on bandwidth limit imposed by network devices. It uses random variables to characterize future bandwidth usage.

2.4 Optimizing Energy Consumption

Kim et al.[13] brings in a new perspective of energy consumption as a metric. It proposes a model for estimating the energy consumption of virtual machines and suggests a vm placement algorithm that provides computing resources to VMs according to energy budget.

2.5 Evaluation

Besides introducing new metrices and mechanisms for vm placement, there has been some research work in evaluating these schemes as well. Lee et al.[15] analyzes resource aggregation, performance degradation, and fairness of resource allocation for four important resource types namely, CPU, Cache, Network and Storage. It concludes that specific behavior is dependent on type of resource and quality of workload. It evaluates different heuristics like FFD-prod, FFDSum, DotProduct and L2 on real and synthetic workloads. Mills et al.[18] proposes a new benchmarking infrastructure for evaluating the performance of different vm placement algorithms.

3

System Overview

Vplacement is virtual machine placement system. This system deploys a VM placement mechanism which optimizes the application response time. It focuses on reducing the resource contention so that overall application performance can be improved. In this chapter we will provide a high level system overview of Vplacement. Following chapters will explain individual system components in detail.

3.1 Goal

The goal of this system is to provide efficient resource utilization while still maintaining required application performance level. This system focuses on getting the efficient VM-PM mapping without any requirement of resizing the virtual machines or introducing new PMs.

3.2 Overview

Figure 3.1 shows vplacement deployment scenario. It contains following components.

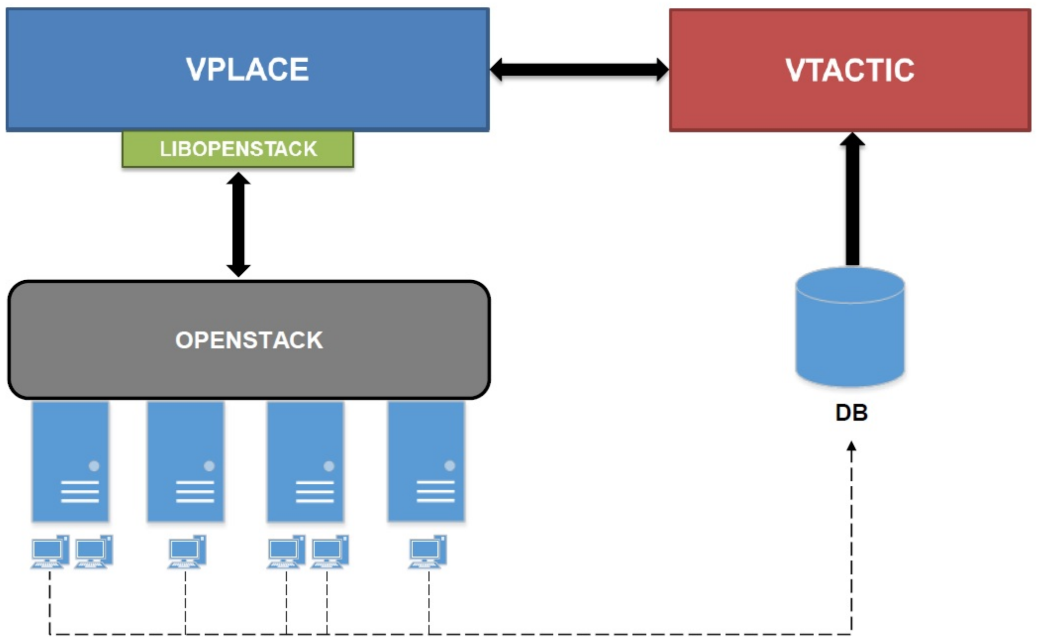


Figure 3.1: Vplacement.

3.2.1 Vtactic

Vtactic [21] is deployed as traffic analyzer. It is the key component of the system. It works on analysis on inter VM traffic. Using the captured traffic it derives the component nodes of an application and the dependencies between them. It estimates the response time of components nodes and application. It further provides an estimate of impact of a component on application performance and estimates the scale of contention using coarrival probability [22].

3.2.2 Openstack

Openstack [3] is used as cloud management system. It takes care of managing pools of compute, storage and network resources.

3.2.3 Libopenstack

C language binding for openstack cloud. It provides get/put interface. It is used by vplacement to interact with openstack.

3.2.4 Vplace

It is the placement engine of the system. It interacts with vtactic and libopenstack to get the combined state of the application VMs. It runs the placement algorithm and migrates the VMs to there respective hosts.

3.3 Working

- VMs capture network traffic and dump it to the database.
- Vtactic analysis the traffic and maintains the probability distribution functions.
- Vplace retrieves the current VM-PM placement data from openstack (using libopenstack).
- Vplace retrieves the impact and coarrival estimates from vtactic.
- Vplace runs the placement algorithm and triggers the VM migrations.

Following sections will cover each vplacement module in detail.

4

Vtactic

Vtactic [21] is an application performance management system. It is used as a traffic analyzer in Vplacement system. It uses the probability of coarrival to quantify the resource contention amongst VMs. It analysis the inter VM traffic to understand how each component impacts the application response time. It uses the analysis to understand as to how much contention a component node can tolerate before it significantly affects the application response time. It proposes a dependency-based application response time modal.

4.1 Dependency Analysis

Vtactic uses dependency primitives to determine the probability distribution of application response time. These primitives are used to understand the execution path of the request.

Vtactic introduced following dependency primitives to classify the request execution flow.

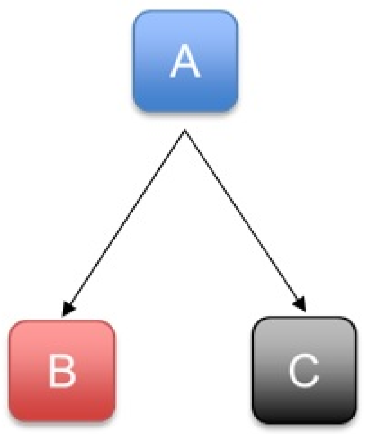


Figure 4.1: Composite Dependency [21].

4.1.1 Composite

Figure 4.1 shows a composite dependency primitive. In composite dependency requests are serially processed. The operator * indicates a composite dependency. For example, A = B * C, signifies that A sends a request to B and after it received a reply from B it sends the request to C.

4.1.2 Concurrent

Figure 4.2 shows a concurrent dependency primitive. In concurrent dependency requests are concurrently processed. The operator || indicates a concurrent dependency. For example, A = B || C, signifies that A sends requests to B and C concurrently.

4.1.3 Distributed

Figure 4.3 shows a distributed dependency primitive. In a distributed dependency request is processed by only one of the child components. The operator + indicates a distributed dependency. For example, A = .2B + .8C, signifies that A sends a

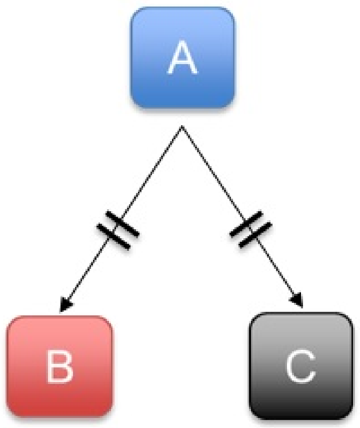


Figure 4.2: Concurrent Dependency [21].

request to either B or either C with probabilities .2 and .8 respectively.

These dependency primitives are used to understand the execution behavior and hence in doing the response time and processing time analysis. For example response time of a component node A is represented as,

$$R_A = S_A + P_A \tag{4.1}$$

Where R_A is the response time of component A, P_A is the processing time of component A and S_A is the response time of subsystem of component A.

If probability distribution of P_A is represented by f(p_a), then in terms of dependency primitive response time can be seen as,

$$f(r_a) = f(s_a) * f(p_a) \tag{4.2}$$

So given the subsystem response time f(s_a) and component's processing time f(p_a), we can derive the probability distribution of component's response time. Similarly

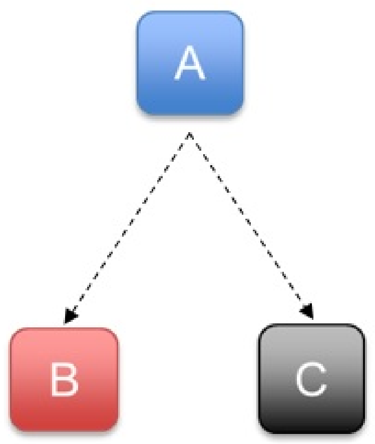


Figure 4.3: Distributed Dependency [21].

given the subsystem response time f(s_a) and component's response time f(r_a), we can derive the probability distribution of component's processing time.

Vtactic evaluates the overall application response time by applying the same concept recursively.

4.2 Impact Analysis

Using the response and processing time analysis mentioned in previous section, Vtactic calculates the impact of a component node on overall application response time. Vtactic uses following steps to evaluate the impact of component A on application performance.

a. Let us assume that current application response time is R_{APP}. b. Right shift the probability distribution of A's processing time f(p_a). c. Evaluate the new response time f'(r_a). d. Reevaluate the application response, we will refer this new application response time as R'_{APP}. e. Now quantified value of impact is represented as,

$$Impact = (R'_{APP} - R_{APP}) \div R_{APP} \tag{4.3}$$

This quantified impact is used by the placement engine to prioritize the VMs.

4.3 Contention Analysis

Another important work done by vtactic is quantifying the degree of resource contention.

4.3.1 CoArrival

Vtactic uses the probability distribution functions of inter-arrival, lag and processing time to verify if 2 requests co-arrive or not. It uses the mechanism mentioned in [22] to estimate if two requests coarrive or not. Two requests for VMs VM1 and VM2, where P_i, R_i and D_i are the processing time, arrival time and deadline of the requests at VM_i will be considered as coarriving if they meet following conditions.

$$R_1 < R_2 + P_2 \tag{4.4}$$

$$min(D_1, D_2) < D_2 < P_1 + P_2. \tag{4.5}$$

Figure 4.4 shows a scenario where VM1 and VM2 coarrive while VM1, VM3 and VM2,VM3 do not.

4.3.2 CoArrival Probability

This coarrival estimation is used to evaluate coarrival probability. CoArrival probability is simply the ratio of number of requests that coarrive and total number requests. It is used as an indicator to represent the resource contention that a VM could be facing.

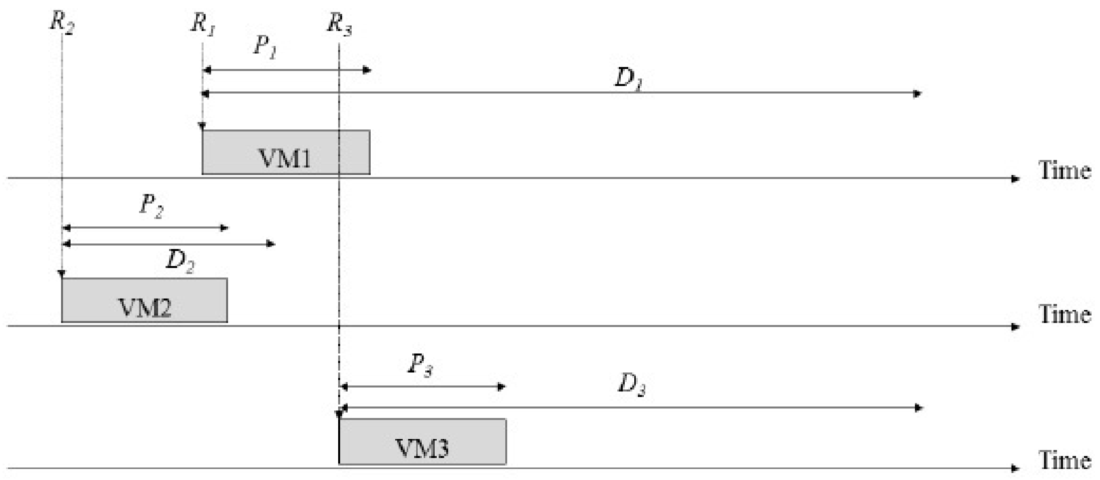


Figure 4.4: Coarrival Probability [22].

4.4 REST APIs

During the vplacement development we have added a REST api interface to vtactic. This enabled vplacement to interact with it. Table 4.1 provides an overview of some important REST apis.

HTTP METHOD	URI	DESCRIPTION
GET	get_model	Gets the vtactic modal for a given application
GET	get_coarrivals	Gets the coarrival probability for a given node
GET	get_app_eval	Gets the estimated response time for a given application
POST	prepare_app	Provides applciation's related input to vtactic
POST	prepare_placement	Provides current VM-HOST Placement mapping

Table 4.1: Vtactic Rest APIs.

5
Openstack

Openstack is a cloud operating system, which manages large pools of compute, network and storage resources which are all managed through a dashboard [3]. Figure 5.1 shows a basic diagram of important openstack modules.

5.1 Compute

Openstack provisions on demand computing resources by provisioning and managing a large network of virtual machines [3]. Openstack nova module is responsible for manging the compute resources.

5.2 Storage

Openstack supports both object and block storage. It provides a distributed storage platform [3]. Openstack modules swift and cinder are responsible for managing object and block storage respectively.

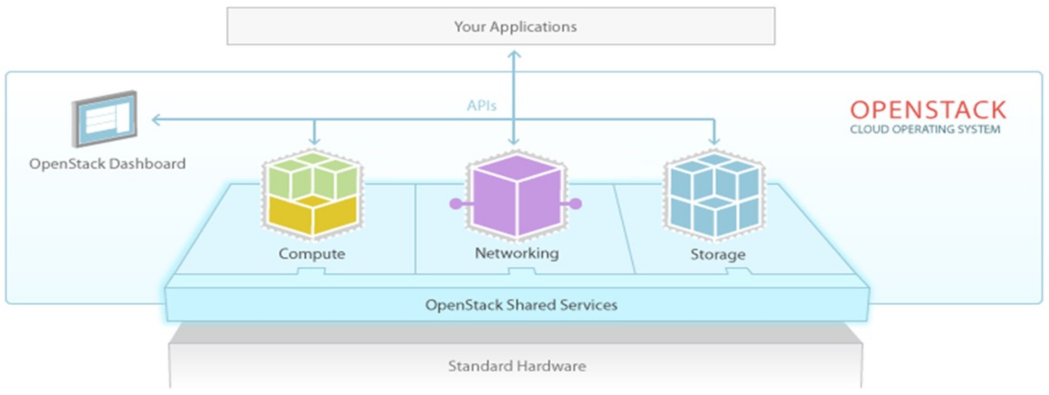


Figure 5.1: Openstack [3].

5.3 Network

Openstack manages the networking services required for compute resources [3]. Nova-network, Neutron (also known as Quantum) are openstack modules which manage the openstack network.

5.4 Dashboard

It provides a graphical interface to access, provision and automate cloud based resources [3].

As a part of this project we explored the nova module of openstack, especially focusing upon the instance launch. Following section explains the design details of instance launch, then we will discuss the concerns related to it and design changes that we tried out.

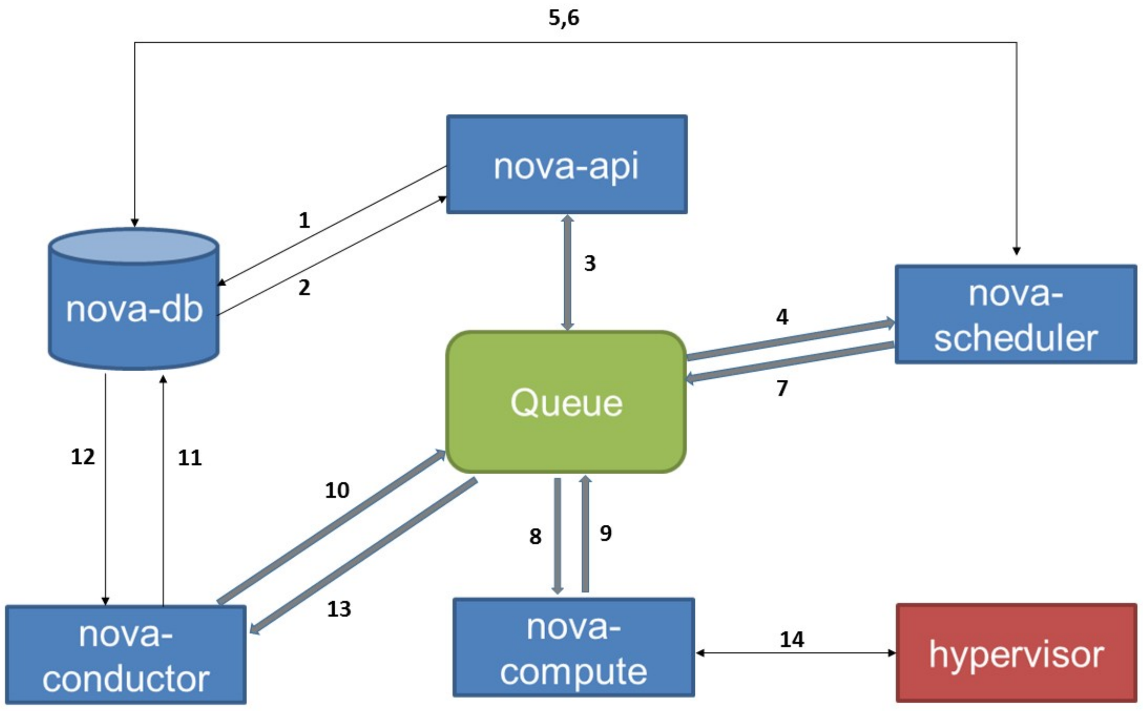


Figure 5.2: Nova instance launch [4].

5.5 Instance Launch

5.5.1 Overview

Figure 5.2 shows the nova modules and interactions that are involved in launching an instance. Please note that we have not shown the interaction with other openstack modules like glance, quantum etc.

Table 5.1 shows a brief overview of nova modules involved [4].

5.5.2 Design Details

Table 5.2 shows basic sequence of events for launching a virtual machine [4]. nova-api is the first module that gets the user request.

5.6 Concerns

One of the major concern with deploying openstack is that it does not support adding a running instance. Consider a scenario where we have a physical host running a few virtual machines. Now, if we want this physical machine to be added to openstack cluster then there is no way i can adding its running VMs to openstack without any downtime.

5.7 Recommended Fix

We analyzed the nova code in detail and tried to fix this problem. Following sections discuss the basic design requirement, concerns with it and the actual design that worked.

5.7.1 Basic Design Requirement

The basic steps that should be followed in order to fix the issue are.

- Nova-compute should query underlying hypervisor for running vms which are not in openstack database.
- Nova-compute should build the data about the running instances with the help of hypervisor.
- Nova-compute should populate the openstack state for the instance (i.e db, meta-data etc.)

5.7.2 Basic Design Requirement (ISSUES)

Following discusses if nova code can support the design requirement.

- Nova-compute should query underlying hypervisor for running vms which are not in openstack database.

Status: Can be easily done.

- Nova-compute should build the data about the running instances with the help of hypervisor.

  Status: Not all the data required by openstack is provided by libvirt. For example, logical network, owner of vm, vm image etc.

- Nova-compute should populate the openstack state for the instance (i.e db, meta-data etc.).

  Status: By design nova compute cannot update the db.

5.8 Improvised Fix

In order to accommodate the limitations (especially about the db update by nova compute), we implemented following design.

The basic design goal is to simulate as if the request for adding the vm came from user. This way we can minimize the changes involved in nova code and hence avoid any unknown scenarios.

Following is the sequence of steps which will be run to add a running vm to openstack.

- Nova-compute will query underlying hypervisor for running vms which are not in openstack database.

- Nova-compute will build the data about the running instances with the help of hypervisor.

- Nova-compute will send a rpc to nova conductor. The rpc will be basically a vm launch request from nova-compute.

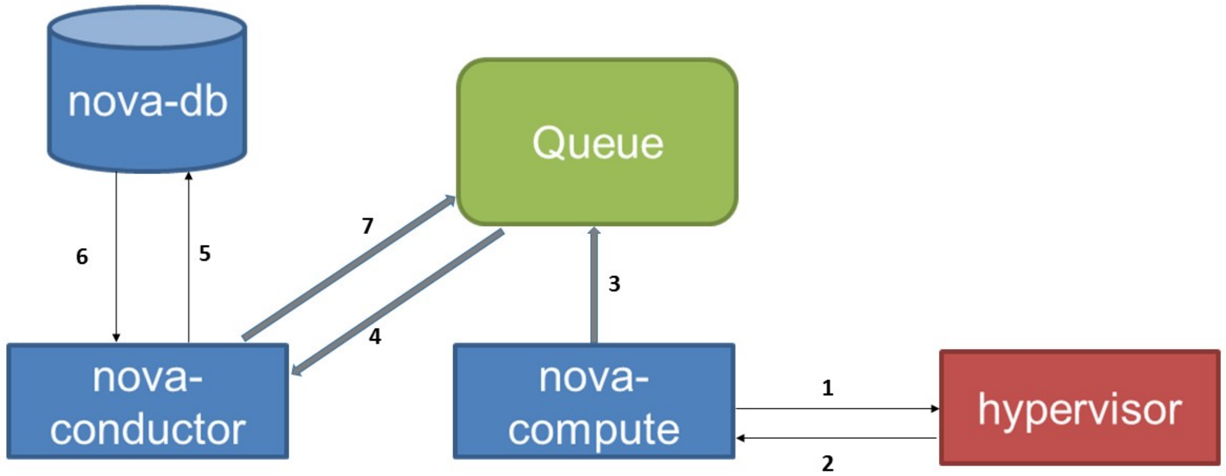


Figure 5.3: Nova improvised fix [4].

- Nova-conductor rpc handler will simulate as if the rpc request as if the instance launch came from user.

Figure 5.3 shows the nova modules and interactions that are involved in implementing the improvised fix.

5.9 Outcome

We implemented the improvised fix in nova module and verified that following operations worked fine.

- Listing the running VMs.
- Stopping the VMs.

- Restarting the VMs.
- Rebooting the VM.

5.10 Future Work

Although the basic vm management operations worked fine with the improvised fix, but providing the networking functionality is still a concern.

Since the libvirt api did not provide the ip address that is configured on the vm, hence it is not possible for openstack to figure out the network vm belongs to.
On Similar lines, the vm could be configured for a different virtual bridge then the one configured in openstack. This information is also not available through libvirt apis.

One of the major task in future will be to analyze the nova-network code and understand if above mentioned scenarios could be fixed.

NOVA MODULE	**DESCRIPTION**
nova-api	handles all the requests related to compute resources
nova-scheduler	decides which compute node instance should be launched
nova-conductor	it is a rpc server, saves nova-compute from accessing database
nova-compute	Provisions instances on hypervisors

Table 5.1: Nova modules involved in instance launch process.

STEP NUMBER	**DESCRIPTION**
1,2	nova-api does db update, indicating the vm state as launching
3,4	nova-api forwards the request to nova-scheduler
5,6	scheduler selects a compute host and updates vm state as scheduled
7	scheduler forwards the request to selected compute node
8,9	nova-compute processes the request and send db update request
10,11,12,13	nova-conductor updates the database on behalf of nova-compute
14	nova-compute send the vm launch request to hypervisor

Table 5.2: Sequence of events in VM launch.

6

Libopenstack

Libopenstack is a static library used by the system to interact with openstack. It provides a simple get/put interface, which can be used by application to read the openstack state or change it. It uses Openstack REST APIs [2].

6.1 Motivation

The motivation for implementing this library was lack of any C binding for openstack interaction.

6.2 Components

Libopenstack uses libcurl [1] and libxml2 [5] libraries.

6.2.1 Libcurl

Used to handle the HTTP connection. Libopenstack uses this library to generate API requests and receiving the response from openstack.

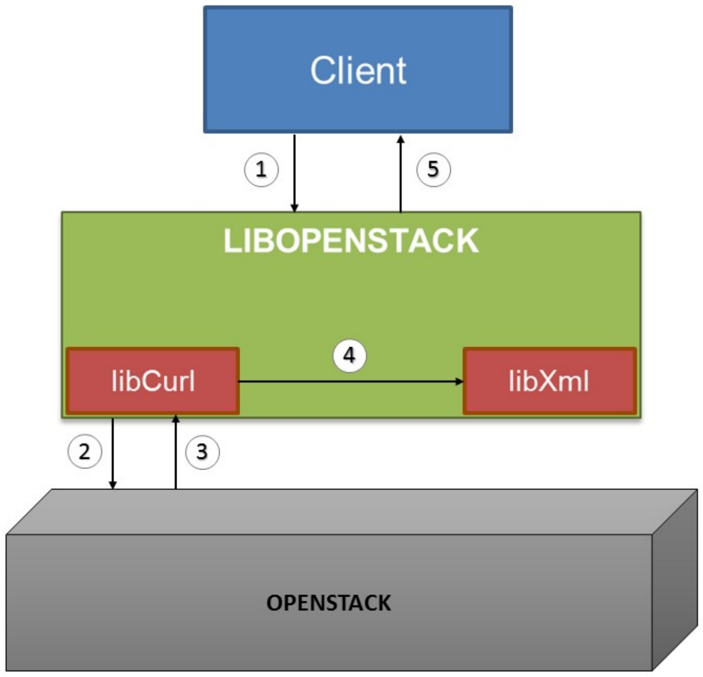


Figure 6.1: Libopenstack.

6.2.2 Libxml2

Used for parsing the XML content. As of now we request openstack to send the response in XML format. Libopenstack uses this library to parse the response returned by openstack.

6.3 Working

6.3.1 Overview

Figure 6.1 gives the overview of libopenstack and handling of a typical application request. Table 6.1 Explains the handling of a typical client request.

6.4 Libopenstack APIs

Libopenstack provides a simple get/put api interface to applications. Till now it supports basic compute APIs but the goal is to extend it further for networking ans storage APIs as well. Table 6.2 provides a brief description of libopenstack APIs available as of now.

STEP NUMBER	DESCRIPTION
1	Application calls libopenstack API.
2	Libopenstack uses libcurl to send HTTP request for corresponding REST API.
3	Libcurl does basic validation of received response.
4	Response body is passed to libxml for parsing.

Table 6.1: Libopenstack Overview.

API	DESCRIPTION
libos_create_session	Authenticates openstack credentials and retrives the authentication token.
libos_get_hosts	Retrieves the information about physical machines in the openstack cluster.
libos_get_instances	Retrieves the information about instances in the openstack cluster.
libos_get_host_instances	Retrieves instances and hosts. It also maps the instances to hosts.
libos_get_flavors	Retrieves the supported VM flavors.
libos_instance_live_migrate	Live migrates input instance to input host.

Table 6.2: Libopenstack APIs.

7

VPlace

Vplace is the placement engine of the system. It interacts with openstack and vtactic to run the placement algorithm, it also provides interface for user interaction. It interacts with openstack to get the current vm placement and for live migrating them. It interacts with vtactic to provide the current vm placement, get the impact and coarrival probabilities. Figure 7.1 provides an overview of components of vplace and there interactions.

7.1 CLI Handler

It is the command shell for user input. Provides commands for run time operations like running the placement algorithm etc.

7.2 Openstack handler

This module handles the interaction with openstack. It uses libopenstack to manage the openstack cluster state. It maintains the following state about the openstack cluster.

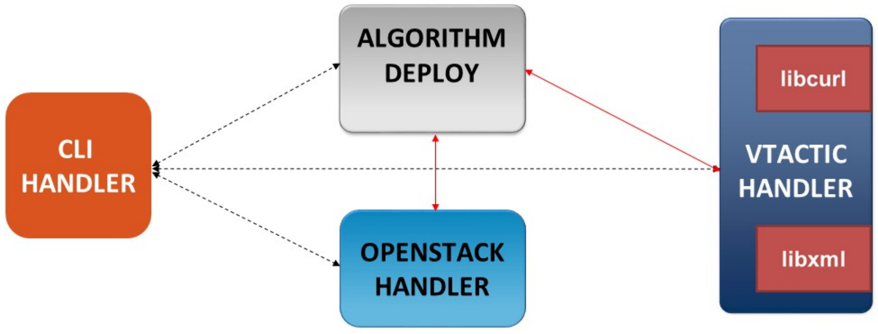


Figure 7.1: Vplace Overview.

session: It saves the authentication token returned by openstack.

Hosts: It manages the list or running physical machines in the cluster.

Instances: It manages the list of instances and mapping between instance and host.

7.3 Vtactic Handler

This module handles the interaction with vtactic REST Apis. It uses libcurl [1] and libxml [5] for generating HTTP requests and parsing the xml reply.

7.4 Algorithm deploy

This module runs the contention aware vm placement algorithm. This modules uses Openstack handler and Vtactic handler to get the vtactic and openstack state and runs the placement algorithm. Following Chapter covers the algorithm in detail.

8
Placement Algorithm

8.1 Goal

Minimize the impact of resource contention on application response time.

8.2 Design Principles

- VM placement is seen as a bin packing problem.
- VMs are prioritized based on there impact on application response time.
- VM-Host mapping is picked to minimize the resource contention for the application.

8.3 Terms and Symbols

Table 8.1 gives an overview of symbols and terms used in the algorithm.

8.4 Concept

The key idea of our placement algorithm is to reduce the resource contention for high impact VMs. We see the VM placement problem as a bin packing problem, where VMs are prioritized based on their impact (higher impact VM is given higher priority) and PMs (bins) are picked based on resource contention that VM will face. The algorithm focuses on reducing the application normalized impact (equation 8.3). When the algorithm is executed for an application (app) it starts picking the component node VMs (higher impact VM first). For each picked VM, algorithm places it on a PM for which AppNImpact_app is minimum. This approach ensures that we gain following.

- Since higher impact VMs are placed first hence they can get the PMs which has minimum contention.
- By having AppNImpact_app as the criteria for placing a VM we ensure that a new VM coming in does not increase the contention on existing VM considerably.

8.4.1 **Example**

Let us say we have an application app with 5 components VM1,VM2,VM3,VM4 and VM5 (ordered by impact) and we have to place them on 3 hosts A,B and C. When we run our algorithm, VM1, VM2 and VM3 will get hosts A, B and C respectively. Now for VM4 and VM5 we should be placing them in such a manner that it should not drastically increase the contention on already placed VMs (VM1, VM2 and VM3). By having the condition of minimizing the value AppNImpact_app we ensure that VM4 and VM5 will go to the PMs for which the the overall contention faced by application is minimum. This ensures that the higher impact VMs which were picked earlier will be cohosted only if the contention caused by new VM coming in does not impact the overall application performance considerably.

An alternate approach is suggested in vtactic [21] as well. The proposed approach is to sort the VMs by NImpact_N. Now pick each VM one by one and place them on a host with minimum coarrival probability. The issues with this approach are that it makes the algorithm dependent on current placement of the VMs and it does not consider that placing a new VM on Host H may increase the resource contention for already placed VM on the same host. In order to analyze our concerns, we evaluate this algorithm as well in our experiments.

Following section covers our algorithm in more detail.

8.5 Algorithm Details

8.5.1 Initial State

Consider following as the initial software state of the system.

- C_A is a set of n component nodes of application A.

$$C_A = \{C_{A1}, C_{A2}, ..., C_{An}\} \tag{8.4}$$

- H is a set of m hosts in which n elements from C_A has to be placed, where m is less then n.

$$H = \{H_1, H_2, ..., H_N\} \tag{8.5}$$

- UC_A is a set of component nodes from C_A to which algorithm has not assigned a host yet. Component nodes in this set are sorted by Impact_N.

$$UC_A = \{C_{A1}, C_{A2}, ..., \} \tag{8.6}$$

- AC_A is a set of component nodes from C_A to which algorithm has assigned a host.

$$AC_A = \{C_{A1}, C_{A2}, ..., \} \tag{8.7}$$

- In the initial state, set AC_A is empty, whereas, set UC_A contains all the nodes from C_A.

- For any component node that belong to UC$_A$ its total coarrival probability is Zero (because it does not belong to any host yet). And hence in the initial state $AppNImpact_A = 0.0$.

8.5.2 Steps

The goal of algorithm is to minimze the impact of resource contention on application response time. We achieve this coming up with a placement scheme the minimizes the value AppNImpact$_A$. Following are the sequence of steps which algorithm runs for each element of set C$_A$ for application A.

1. Pick first element C$_{Ai}$ from set UC$_A$.
2. For each host H$_i$ from the set H, assign C$_{Ai}$ to H$_i$ and calculate AppNImpact$_A$.
3. Pick the host H$_{min}$ for which AppNImpact$_A$ value was minimum. Assign C$_{Ai}$ to H$_{min}$ and move it to set AC$_A$.
4. Move to step 1. if UC$_A$ is not empty.
5. Figure 8.1 gives an overview of algorithm steps.

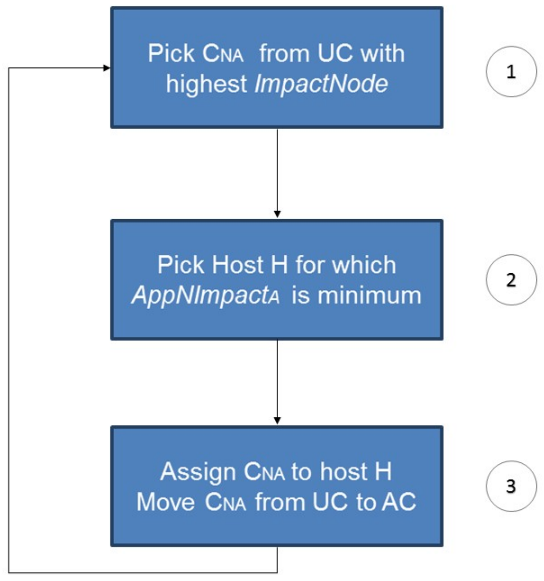

Pick CNA from UC with highest *ImpactNode*
Pick Host H for which *AppNImpactA* is minimum
Assign CNA to host H Move CNA from UC to AC
1
2
3

Figure 8.1: Placement Algorithm Steps.

TERM	**SYMBOL**	**DESCRIPTION**
Impact	Impact_N	Impact of Component Node N on application response time.
CoArrival Probability	$\text{coArr}_{N1,N2}$	Probability that Node N1 processes requests at the same time as Node N2.
Total CoArrival Probability	TcoArr_{N1}	Summation of coarrival probabilities of Node N1 with all the other instances on Host H, where H is the Host on which Node N1 is placed. $$TcoArr_{N1} = \sum_{i=2}^{M} coArr_{N1,Ni} \quad (8.1)$$ Where M is the number of instances on Host H (excluding Node N1).
Normalized Impact	NImpact_N	Product of impact and total coarrival probability of Node N. $$NImpact_N = Impact_N * TcoArr_{N1} \quad (8.2)$$ Where H is the host where Node N is currently placed.
App Normalized Impact	AppNImpact_{app}	Summation of normalized impacts of all component nodesi of application app. $$AppNImpact_{app} = \sum_{i=1}^{M} NImpact_{Ni} \quad (8.3)$$ Where M is the number of component nodes for application app.

Table 8.1: Terms and Symbols involved in placement algorithm.

9

Experiments and Results

This chapter covers the experiments we conducted for evaluating the performance of our placement algorithm. All the experiments were conducted using the vplacement system.

9.1 Overview

The focus of our experiments is to verify that given an initial placement and restriction of not to resize VMs or add new PMs, our placement algorithm improves the application response time by finding the appropriate colocated VMs.We have created a simple web applciation with 5 nodes. As a part of request processing each node runs sha hashing on a 2 GB file. We vary the load on each VM by changing the number of iterations of sha calculation. For example in Figure 9.1, the value under the VM name indicates the number of iterations of sha calculation for a request processing. On each PM we keep only one cpu core as enabled, this is done to ensure that there is a cpu resource contention and to show that in case of resource contention our placement algorithm can be effective.

We use the initial placement of application as the base result, i.e we use the simple round robin scheduler by openstack to place the application initially, and the performance of application with this placement is taken as the base result. We use the base result to measure the improvement achieved by our algorithm.

We use the distribution of application response time as the metric for application performance. For a given input workload, we measure the number of responses received within certain time interval.

We ran three experiments. Each experiment focused on a dependency primitive (Composite, Concurrent and Distributed). Following sections provide details about the experiments conducted and the observed results.

9.2 Setup

9.2.1 Physical Machines

Three physical machines Host A, Host B and Host C. Table 9.1 provides an overview of host configurations.

9.2.2 Virtual Machines

Five Virtual machines VM1, VM2, VM3, VM4 and VM5. Table 9.2 provides an overview of virtual machine configurations.

9.2.3 Openstack

We used openstack havana release. Table 9.3 shows the software versions of critical services running in the openstack cluster.

9.3 Profiling

- The first step towards using the vplacement is to do profiling.
- By profiling we mean that we should deploy the application such that there is no resource contention. For example, in this experiment we will deploy the 5

component nodes (VM1, VM2, VM3, VM4 and VM5) of the application on 5 different hosts (we added 2 more hosts hostD and hostE for profiling purposes only).

- On each virtual machine we started wireshark to capture traffic.
- Once application is deployed then we run the desired traffic load.
- Captured traffic on virtual machines is transferred to a database.
- This database is used by Vtactic [21] for traffic analysis.

9.4 Algorithms

Table 9.4 gives an overview of the placement algorithms we run for evaluation. VPLACE_4 is the algorithm that we have discussed in section 8. We compare our placement algorithm with VPLACE_2 to indicate that a contention unaware placement algorithm may not give the expected results. We compare VPLACE_4 with VPLACE_5 as well, this is done to indicate the VPLACE_5 is dependent on initial placement of VMs and it may still lead to a high resource contention.

9.5 Experiment 1 (COMPOSITE)

9.5.1 Topology

- In this experiment we tested a 5 node (VM1, VM2, VM3, VM4 and VM5) composite topology application.
- Each Node computes sha of a 2 GB file.
- The sha computation is done in a loop, the number of iterations will vary on each node.
- Figure 9.1 shows the sequence of interactions for processing a HTTP request.

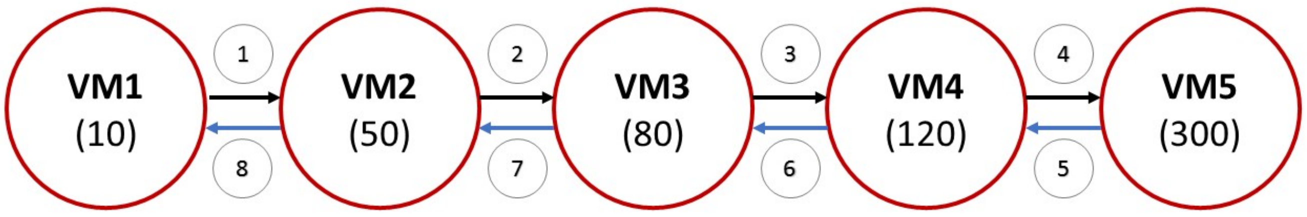


Figure 9.1: Experiment 1 topology.

9.5.2 Initial Placement

Figure 9.2 shows the initial placement of virtual machines.

9.5.3 Traffic

- 100 parallel TCP connections.

- 10 requests per second.

9.5.4 Node Impact

VM1: 1.392466

VM2: 1.949453

VM3: .194132

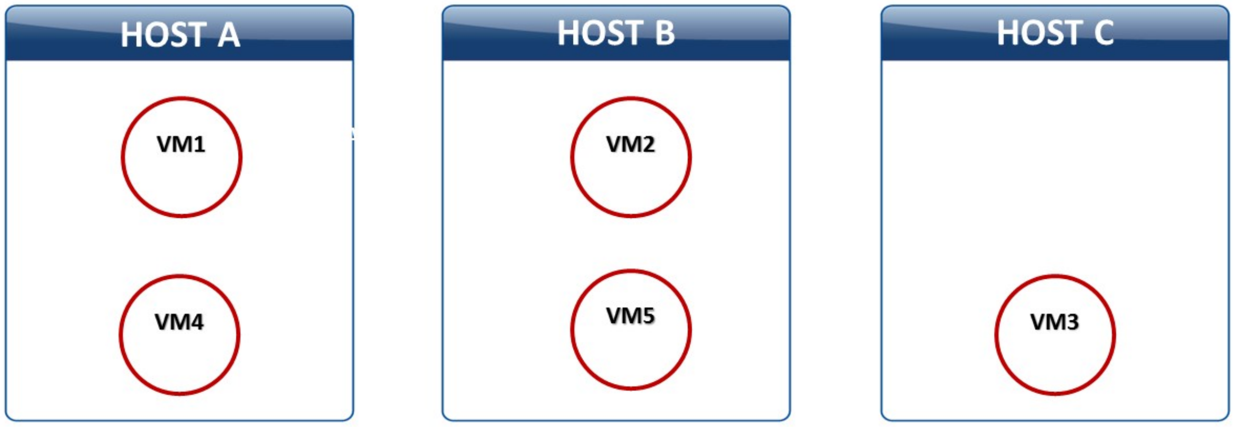


Figure 9.2: Experiment 1 initial placement.

VM4: .410988

VM5: .303830

9.5.5 **Result**

Figure 9.3 shows the final VM placement for each algorithm. Following is the analysis for final placement.

- As the figure 9.3 shows, VPLACE4 colocated VMs VM3,VM4 and VM5. As per the algorithm, it sorted VMs by their impact on application response time. Following is the sequence of VMs sorted by priority (higher priority VM first), VM2, VM1, VM4, VM5, VM3. After placing VM2,VM1 and VM4 on hosts Host B, Host C and Host A respectively, it decided to place VM5, VM3 on

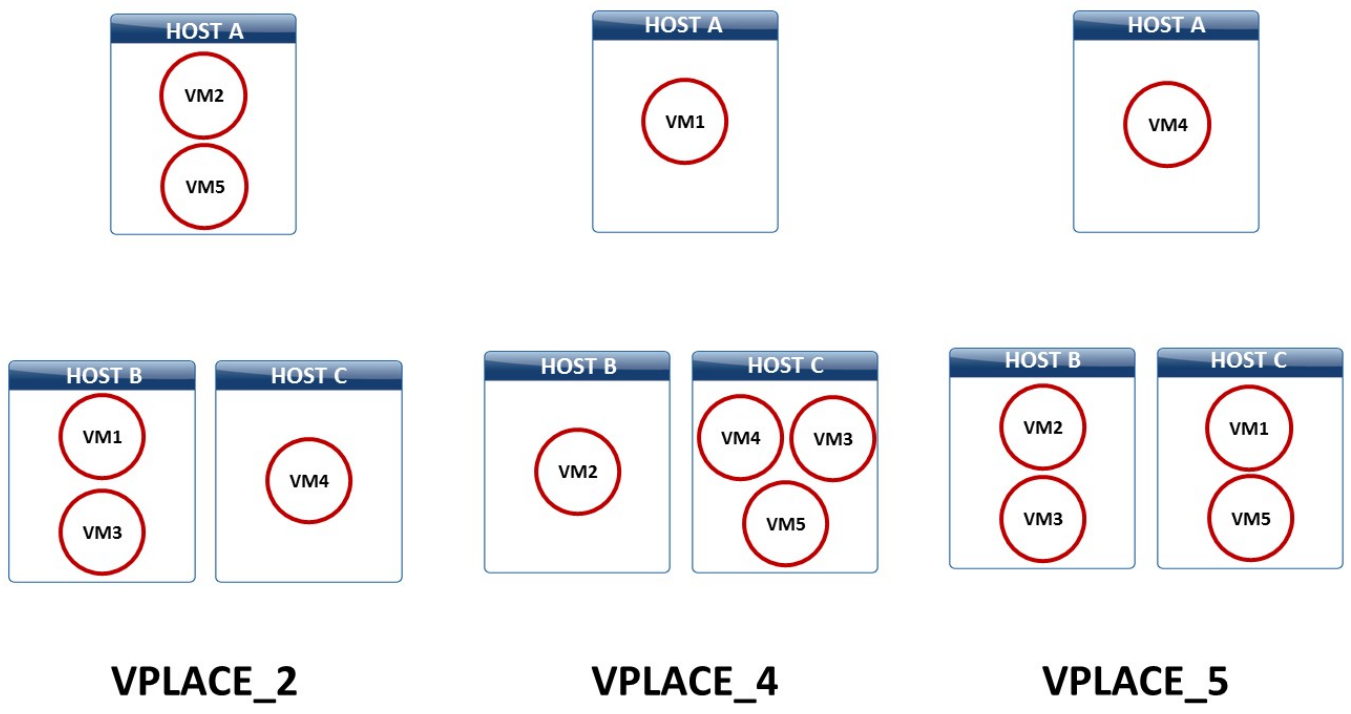


Figure 9.3: Experiment 1 placement results.

Host A. This was done because the impact of VM3, VM4 and VM5 is considerably lesser then other VMs, hence placing them together lead to minimum AppNImpact_app value.

- VPLACE2 final placement result is simply a result of WFD bin packing heuristic.
- VPLACE5 here sorted the VMs by NImpact_N and placed VM2,VM1 and VM4 on each host. But since it did not take into consideration the coarrival probability of already placed VMs, hence it ended up placing VM3 with VM2 and VM1 with VM5.

Figure 9.4 shows the distribution of response time. Where X axis is response time, and Y axis shows the number of responses. It also shows the percentage of requests for which response was received. Following is the analysis for the results observed.

- As shown in Figure 9.4, placement governed by VPLACE4 lead to a much better application response time then the base distribution. The maximum response time was 51 seconds and most of the requests had lower response time values as compared with placement achieved by other algorithms. This happened because it reduced the contention on high impact VMs (VM1, VM2). It increased the contention on VMs VM3, VM4 and VM5 by placing them on same host, but because of there low impact the application performance did not degrade much.

- For VPLACE2 and VPLACE5 the performance actually degraded then the base. In fact, we did not receive responses for all the requests. This happened because placements decided by both the algorithms increased contention on VM1 and VM2.

- The average response time of the application came down from 53.97 seconds (Base) to 37.12 seconds (VPLACE4).

9.6 Experiment 2 (DISTRIBUTED)

9.6.1 Topology

- In this experiment we tested a 5 node (VM1, VM2, VM3, VM4 and VM5) distributed topology application.

- Each Node computes sha of a 2 GB file.

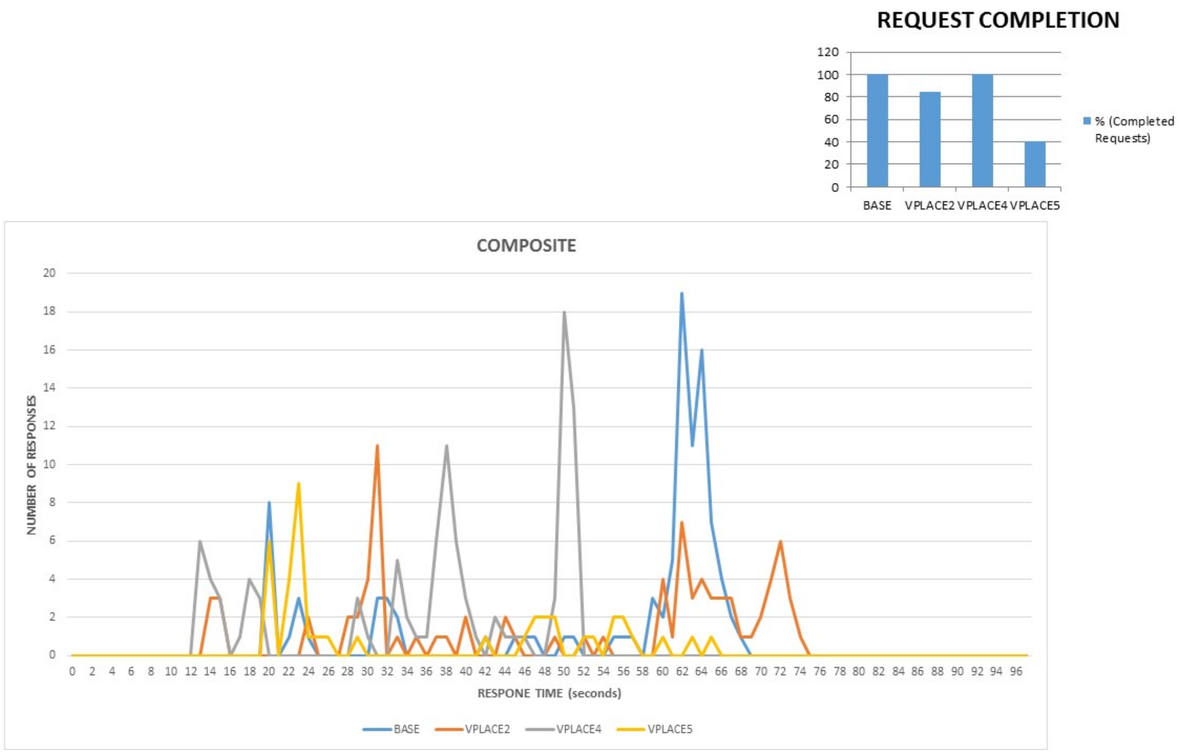


Figure 9.4: Experiment 1 placement evaluation results.

- The sha computation is done in a loop, the number of iterations will vary on each node.
- Figure 9.5 shows the sequence of interactions for processing a HTTP request. As shown in the figure VM1 forwards the request to either VM2 or VM4 with probabilities .3 and .7 respectively.

9.6.2 Initial Placement

Figure 9.6 shows the initial placement of virtual machines.

9.6.3 Traffic

- 200 parallel TCP connections.

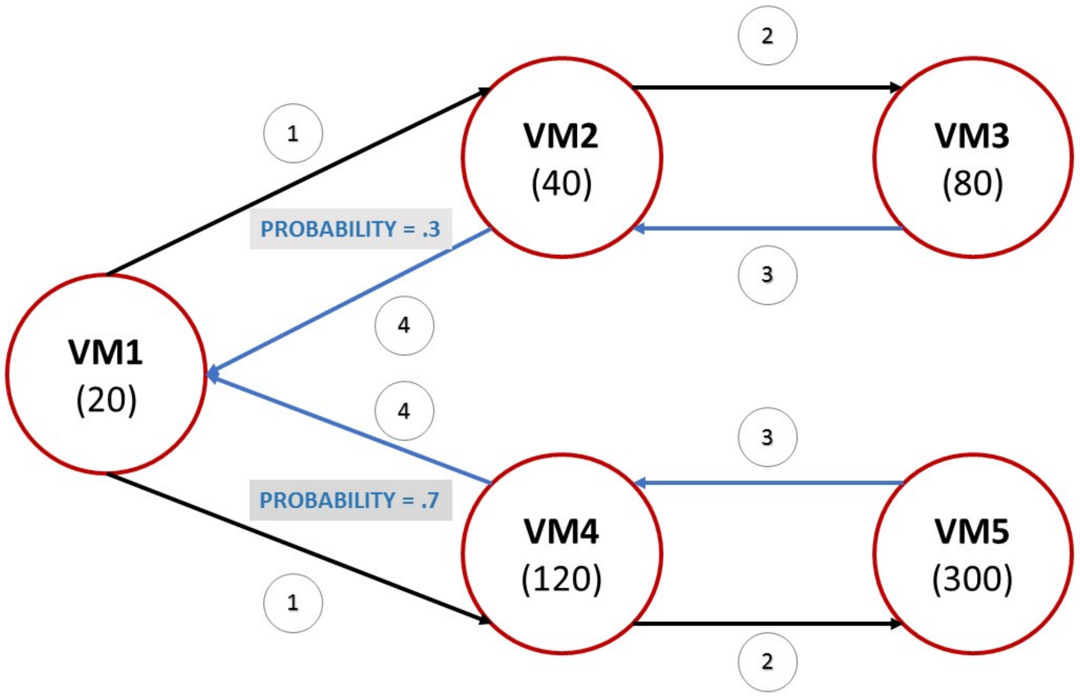


Figure 9.5: Experiment 2 topology.

- 20 requests per second.

9.6.4 Node Impact

VM1: 2.560027

VM2: .084937

VM3: .056401

VM4: .310149

VM5: 1.124396

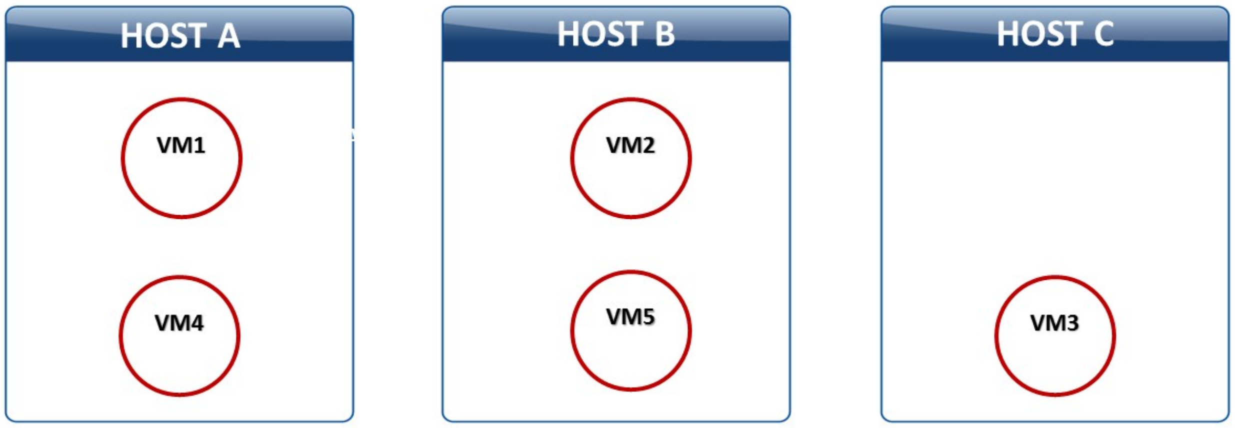


Figure 9.6: Experiment 2 initial placement.

9.6.5 Result

Figure 9.7 shows the final VM placement for each algorithm. Following is the analysis for final placement.

- As the figure 9.7 shows, VPLACE4 colocated lowest impact VMs VM2,VM3 and VM4. As per the algorithm, it sorted VMs by their impact on application response time. Following is the sequence of VMs sorted by priority (higher priority VM first), VM1, VM5, VM4, VM2, VM3. This time VMs VM2 and VM3 has very low impact because as per our topology only 30 percentage of traffic if forwarded to them. Hence our placement algorithm decided to cohost VM2, VM3 and VM4. Allowing VM1 and VM5 (high impact VMs) to work without contention.

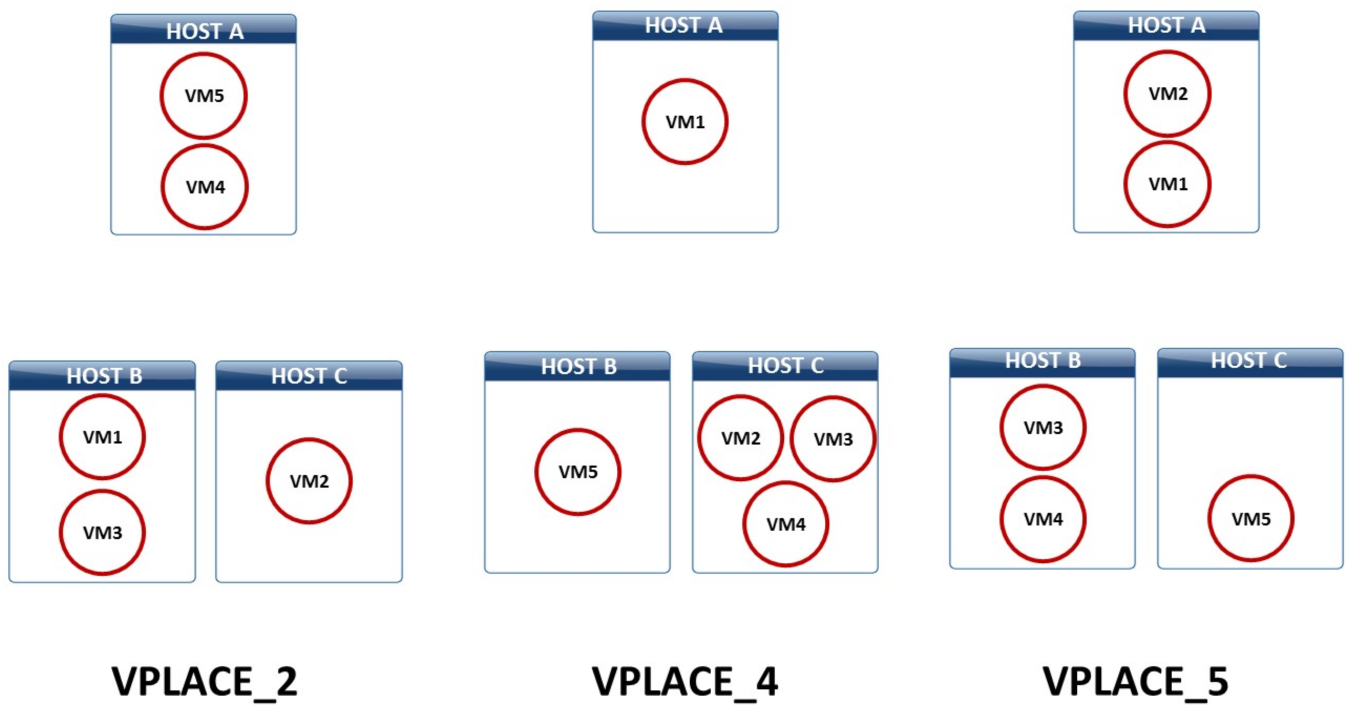


Figure 9.7: Experiment 2 placement results.

- VPLACE2 final placement result is simply a result of WFD bin packing heuristic.
- VPLACE5 did prioritize the VMs VM1 and VM5 but while placing other VMs it increased contention on VM1.

Figure 9.8 shows the distribution of response time. Where X axis is response time, and Y axis shows the number of responses. It also shows the percentage of requests for which response was received. Following is the analysis for the results observed.

- As shown in Figure 9.8, placement governed by VPLACE4 lead to a much better application response time then the base distribution. This happened because placement by VPLACE4 reduced the contention on high impact VMs. It increased the contention on VMs VM2, VM3 and VM4, but because of their significantly low impact, the application performance did not degrade much.

- For VPLACE2 the performance actually degraded then the base. In fact, we did not receive responses for all the requests. This happened because VPLACE2 colocated VM4 and VM5. Since both the VMS face 70 percentage of the traffic hence they have high impacts and higher coarrival as well. Hence because of resource contention the application performance degraded by quite a bit.

- VPLACE5 also improved application response time. But its performance was still lower then VPLACE4. The reason for good performance for VPLACE5 is that although it colocated a high impact VM (VM1) with VM2, but since VM2 faces only 30 percentage of traffic hence both its coarrival probability and impact is low, not affecting the overall application response time much.

- The average response time of application came down from 64.365 seconds to 57.855 for VPLACE5. Where as it came down to 38.745 for VPLACE4.

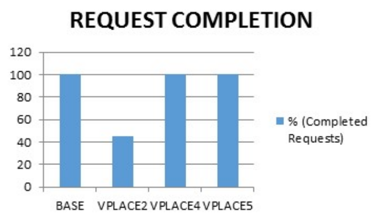
9.7 Experiment 3 (CONCURRENT)

9.7.1 Topology

- In this experiment we tested a 5 node (VM1, VM2, VM3, VM4 and VM5) concurrent topology application.

- Each Node computes sha of a 2 GB file.

- The sha computation is done in a loop, the number of iterations will vary on each node.



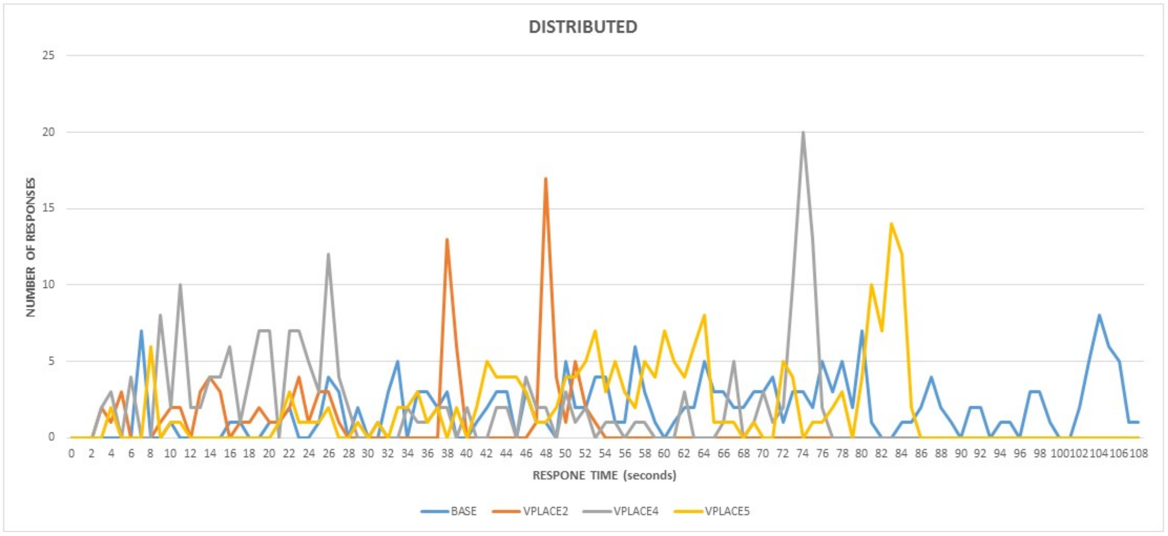


Figure 9.8: Experiment 2 placement evaluation results.

- Figure 9.9 shows the sequence of interactions for processing a HTTP request. As shown in the figure VM1 forwards the request to both VM2 and VM4.
- One point to observe about this dependency is that there is a lot of parallel processing in the system, i.e for example as we observed in composite dependency, VM1 forwarded the request to VM2 which forwarded it further, i.e the request were being processed sequentially there. For distributed dependency since we pick either one path or another hence for each request only 3 nodes were active at a time. But for concurrent dependency since VM1 forwards the request to both VM2 and VM4 hence both the subsystems process the request in parallel. As a result the load on PMs is much higher with concurrent dependency as compared with other two.

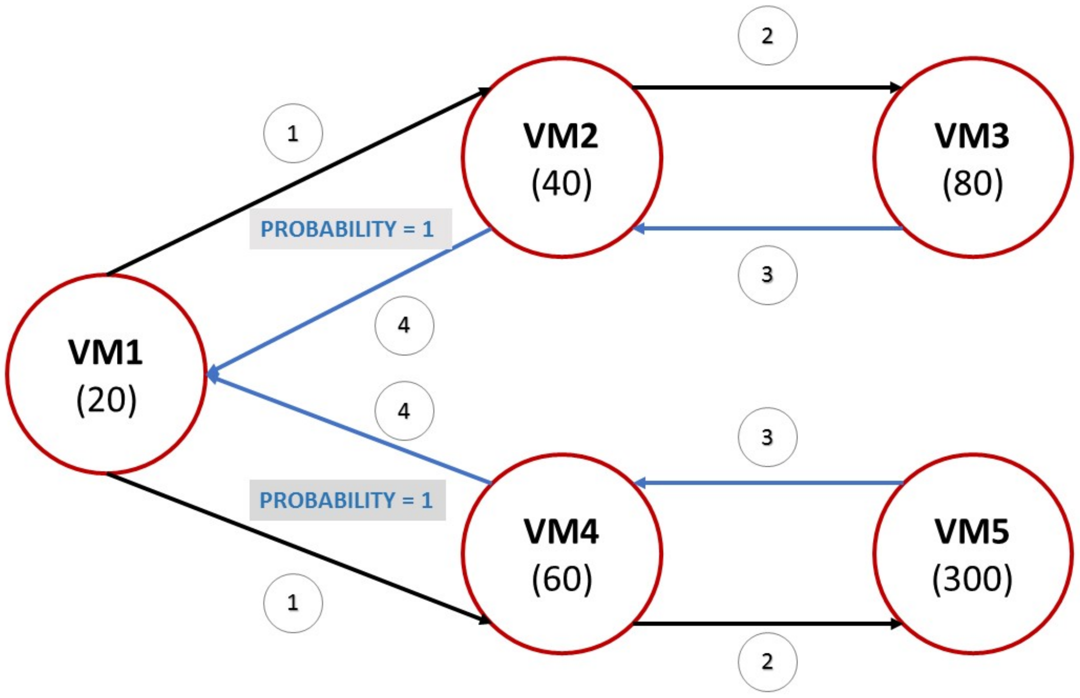


Figure 9.9: Experiment 3 topology.

9.7.2 Initial Placement

Figure 9.10 shows the initial placement of virtual machines.

9.7.3 Traffic

- 110 parallel TCP connections.
- 10 requests per second.

9.7.4 Node Impact

VM1: 2.700713

VM2: .352752

VM3: .058637

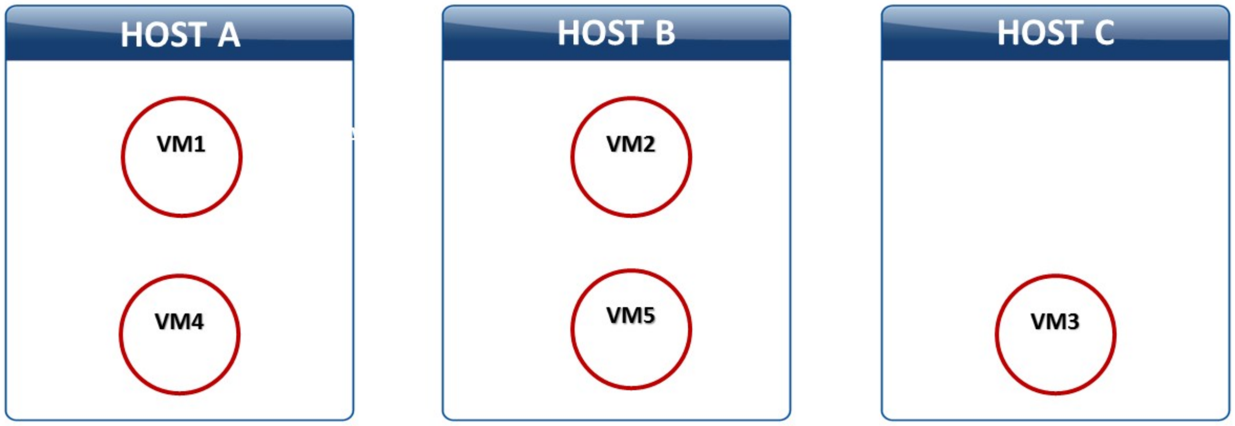


Figure 9.10: Experiment 3 initial placement.

VM4: .621745

VM5: .517421

9.7.5 Result

Figure 9.11 shows the final VM placement for each algorithm. Following is the analysis for final placement.

- Figure 9.11 shows the final placement result by each algorithm.
- This time VPLACE4 gave a different final result, i.e for previous two dependencies we observed that it placed high impact VMs on 2 hosts and colocated 3 lower impact VMs together. But in case of concurrent dependency all the VMs are active at the same time for processing a web request, this leads to a

much higher value of coarrival probability. Hence placing 3 low impact VMs (VM2, VM3 and VM5) did not lead to minimum AppNImpact_app value. As a result we had to colocate VM3, VM4 together and VM2, VM5 together. We still kept VM1 (highest impact VM) alone.

- VPLACE2 final placement result is simply a result of WFD bin packing heuristic.

- VPLACE5 followed the same policy as discussed earlier, it did prioritize the high impact VMS but kept VM4 with VM2 because it did not consider that adding VM4 on Host A increased contention on VM2. One more point worthy of observation here is that this time VPLACE5 prioritized VM2 (impact = .352752) and VM5 (impact = .517421) over VM4 (impact = .621745), this happened because in initial placement VM2 and VM5 were on same host, now since the request is processed in parallel hence VM2 and VM5 have high mutual coarrival probability which makes NImpact_VM2 and NImpact_VM5 much higher then that of VM4. This observation helps us conclude that VPLACE5 is also dependent on initial placement of the component nodes.

Figure 9.12 shows the distribution of response time. Where X axis is response time, and Y axis shows the number of responses. It also shows the percentage of requests for which response was received. Following is the analysis for the results observed.

- In case of concurrent dependency the load on the system is significantly higher then other dependencies. As a result even our base line placement could not process all the requests.

- Placement governed by VPLACE4 processed all the requests.

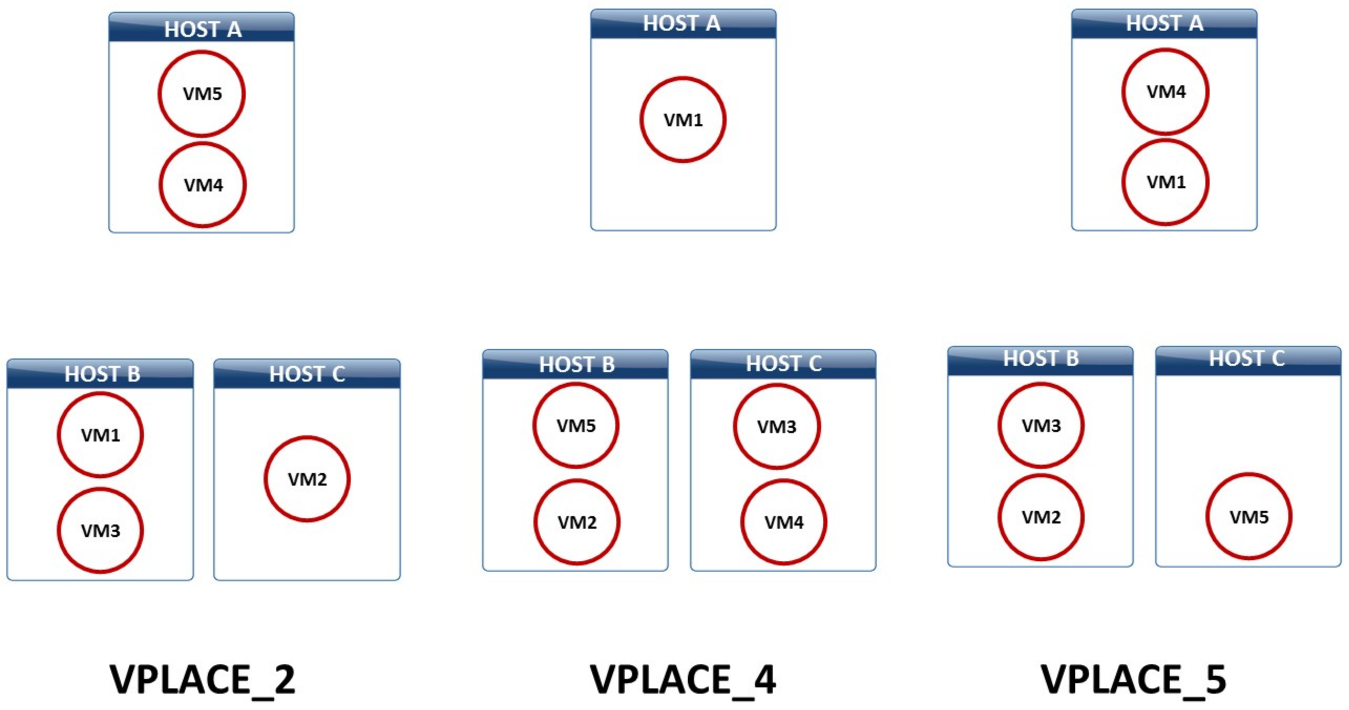
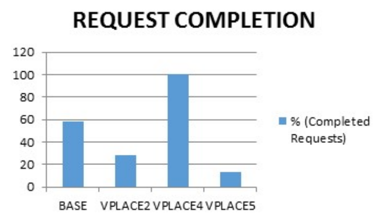


Figure 9.11: Experiment 3 placement results.

- Since the resource demand is much higher now, hence the degradation caused by suboptimal placement are also significantly high.
- As we can see in figure 9.12, placement derived by VPLACE2 and VPLACE5 could only process 28.18 and 13.6 percent of requests respectively. This happened because VPLACE2 placed VM4 and VM5 together, both were high impact VMs. Whereas VPLACE5 placed high impact VMs VM1 and VM4 together.

With the help of our experiments we can conclude that our placement algorithm lead to significant improvement of application response time when application was placed with a simple round robin placement mechanism initially. Our algorithm performed much better then VPLACE2 and VPLACE5. Our experiments also explain



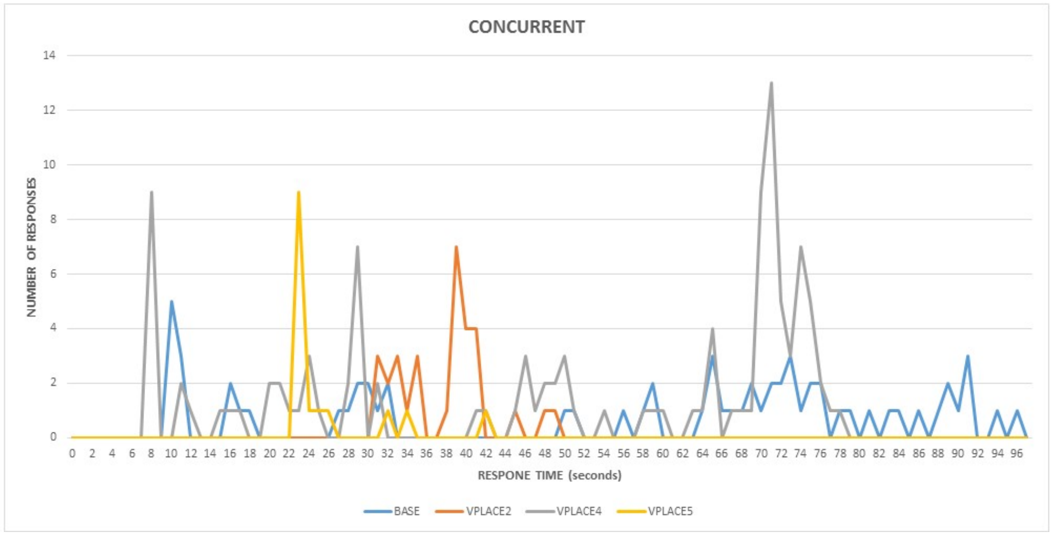


Figure 9.12: Experiment 3 placement evaluation results.

the importance resource contention has on overall application response time.

HOST	MEMORY (GB)	DISK (GB)	CPU (GHz)
A	16	500	3.6
B	16	500	3.6
C	16	500	3.6

Table 9.1: Configuration of physical machines used in the experiments.

VM	MEMORY (GB)	DISK (GB)	Number of Vcpus
1	2	20	1
2	2	20	1
3	2	20	1
4	2	20	1
5	2	20	1

Table 9.2: Configuration of virtual machines used in the experiments.

SERVICE	VERSION
nova-compute	2014.1
nova-scheduler	2014.1
nova-network	2014.1
glance	0.12.0.78

Table 9.3: Versions of Nova and Glance services running in the openstack cluster.

NAME	**DESCRIPTION**
VPLACE_2	• Sort the nodes in decrementing order of their resource requirement. • Sort the hosts in decrementing order of available physical resources. • Use Worst Fit Decrementing bin packing heuristics.
VPLACE_4 [section 8]	• **Sort the nodes in decrementing order of their impact on application.** • **Place the nodes on physical host so that application normalized impact is minimized.**
VPLACE_5 [21]	• Sort the nodes in decrementing order of their normalized impact. • For each node sort hosts in decrementing order of coArrival probabilities. • Use First Fit Decrementing bin packing heuristics.

Table 9.4: Placement algorithms used in evaluation.

10

Conclusion

Achieving a high consolidation ratio while still maintaining desired performance level is a critical task in the cloud. This decision of hosting a virtual machine on a particular physical machine is done by VM placement algorithms. There has been a lot of study in this domain and the algorithms try to optimize various performance metrices. [26] Optimizes the VM response time by monitoring the resource usage. [17] Optimizes the network traffic by cohosting the virtual machines which interact with each other a lot. [13] Proposes a mechanism to measure energy consumption and place virtual machines according to energy budget. Most of the VM response time optimization based placement algorithms focus on resource usage of virtual machines. They measure the resource utilization of (all or some of) cpu, memory, disk and network resources, and they either allocate more resources to VMs or move them to a server which can fulfill the resource requirement. This resource based resizing/placement decisions may lead to inefficient utilization pf physical resources. [12] Demonstrates that the measurement of cpu utilization when under resource contention may give incorrect estimates, it mentions that provided estimate could indicate a much higher resource requirement then what is needed.

In this study we focused on improving a multi-tiered application's response time by minimizing the effect of resource contention. Large scale applications are deployed as multi-tier applications. They use multiple component virtual machines each doing a specific task. We proposed a contention aware VM placement system termed as Vplacement. This system analyzed the dependencies between application components and proposed a placement algorithm which minimizes the impact of resource contention on the application response time. Our placement does not resize the virtual machines, it tries to find the VM placement under the constraint that neither more physical servers should be introduced nor the virtual machines should be resized.

We compare the performance of this algorithm with a simple worst fit decrementing heuristic and another contention aware algorithm proposed in [21]. Our experiments cover the scenarios for different dependency primitives [21]. Our results prove that the placement scheme outperforms both the algorithms. Our placement system does not depend on initial placement of component nodes and is only tied with the traffic pattern that the application is facing.

Bibliography

[1] "libcurl - the multiprotocol file transfer library." [Online]. Available: http://curl.haxx.se/libcurl/." [Accessed January 01, 2014].

[2] "Openstack api documentation." [Online]. Available: http://api.openstack.org/api-ref.html." [Accessed January 01, 2014].

[3] "Openstack open source cloud computing software." [Online]. Available: http://www.openstack.org/." [Accessed July 25, 2013].

[4] "Request flow for provisioning instance in openstack." [Online]. Available: http://ilearnstack.com/2013/04/26/request-flow-for-provisioning-instance-in-openstack/." [Accessed September 25, 2013].

[5] "The xml c parser and toolkit of gnome." [Online]. Available: http://xmlsoft.org/." [Accessed January 01, 2014].

[6] U. Bellur, C. S. Rao, and M. K. SD, "Optimal placement algorithms for virtual machines," *CoRR*, vol. abs/1011.5064, 2010.

[7] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, May 2007, pp. 119–128.

[8] N. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti, "Vm placement strategies for cloud scenarios," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, June 2012, pp. 852–859.

[9] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '03. New York, NY, USA: ACM, 2003, pp. 300–301. [Online]. Available: http://doi.acm.org/10.1145/781027.781067. [Accessed January 25, 2014].

[10] C. Clark, K. Fraser, S. H, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI*, 2005, pp. 273–286.

[11] A. Corradi, M. Fanelli, and L. Foschini, "Vm consolidation: A real case based on openstack cloud," *Future Gener. Comput. Syst.*, vol. 32, pp. 118–127, Mar. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.05.012. [Accessed January 25, 2014].

[12] C. Isci, J. Hanson, I. Whalley, M. Steinder, and J. Kephart, "Runtime demand estimation for effective dynamic resource management," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, April 2010, pp. 381–388.

[13] N. Kim, J. Cho, and E. Seo, "Energy-credit scheduler: An energy-aware virtual machine scheduler for cloud systems," *Future Gener. Comput. Syst.*, vol. 32, pp. 128–137, Mar. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.05.019. [Accessed January 25, 2014].

[14] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, April 2007, pp. 200–209.

[15] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating heuristics for virtual machines consolidation," *Microsoft Research, MSRTR-2011-9*, 2011.

[16] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 248–259. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155650. [Accessed January 25, 2014].

[17] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[18] K. Mills, J. Filliben, and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 91–98.

[Online]. Available: http://dx.doi.org/10.1109/CloudCom.2011.22. [Accessed January 25, 2014].

[19] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 237–250. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755938. [Accessed January 25, 2014].

[20] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European Conference on Computer Systems*, ser. EuroSys '09. New York, NY, USA: ACM, 2009, pp. 13–26. [Online]. Available: http://doi.acm.org/10.1145/1519065.1519068. [Accessed January 25, 2014].

[21] A. Sangpetch, "Tactic: Traffic aware cloud for tiered infrastructure consolidation," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2013, aAI3538969.

[22] O. Sukwong, A. Sangpetch, and H. Kim, "Sageshift: Managing slas for highly consolidated cloud," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 208–216.

[23] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 331–340. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242618. [Accessed January 25, 2014].

[24] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 283–294, Jun. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024723.2000099. [Accessed January 25, 2014].

[25] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 71–75.

[26] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proceedings of the 4th USENIX Conference on Networked Systems Design &#38; Implementation*,

ser. NSDI'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 17–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=1973430.1973447. [Accessed January 25, 2014].