UNIVERSITY OF AMSTERDAM

Master Thesis Report

Improving Quality of Experience of Internet Applications by Dynamic Placement of Application Components

By

Deepthi Devaki Akkoorath

Supervisors: Rudolf Strijkers, Dr. Oskar van Deventer

July 2012

This thesis is submitted in partial fulfillment of the requirement of
Master of Science in Grid Computing
in the
Faculty of Science, Informatics Institute
University of Amsterdam

Title:	Improving Quality of Experience of Internet Applications by Dynamic Placement of Application Components
Author:	Deepthi Devaki Akkoorath
Student number :	10033211
MSc program:	Grid Computing
Track:	Computer Science
Institute:	University of Amsterdam, the Netherlands
Faculty:	Faculty of Science, Informatics Institute
Company:	TNO
Address:	Brassersplein 2 NL-2612 CT Delft
Supervisors	Rudolf Strijkers Dr. Oskar van Deventer
Board of Examiners:	Rudolf Strijkers Dr. Oskar van Deventer Dr. Adam Belloum Dr. Clemens Grelck

Abstract

The Quality of Experience(QoE) of distributed internet applications can be improved by placing servers closer to the clients. In this thesis, we analyse and quantify the impact of application component placement on the QoE. To determine optimal application component placement we propose a metric Communication Affinity. Communication Affinity indicates the coupling between two application components (stretch). Moreover, the presented concepts lead to a novel approach to schedule the virtual machines on distributed clouds.

This work will be submitted to FGCS as:
Deepthi Devaki Akkoorath, Rudolf Strijkers, Marc X. Makkes, Oskar van Deventer, Adam Belloum, Cees de laat, Robert J. Meijer, "A metric for QoE-optimal placement of applications"

Acknowledgements

I would like to thank all those great people without whose help and support, this thesis would not have become the way it is now. First and foremost, I would like to thank my supervisor, Rudolf Strijkers for his continuous support throughout the work. The regular discussions with Oskar van Deventer, Robert Meijer, Marc Makkes and Adam Belloum has been very helpful in my progress. I thank them all for their support.

I would like to thank TNO for taking me as an intern. I am very grateful to Nuffic for supporting me with HSP Huygens scholarship without which my master studies at UvA would not have been possible.

Next, is my friends who had always been my motivation through out my studies. I thank them for proof reading my thesis and for the useful comments which helped me to shape my thesis.

Last but not the least, I thank my parents.

Contents

List of Figures

List of Abbreviations

CDN	Content Delivery Networks
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IP	Internet Protocol
QoE	Quality of Experience
QoS	Quality of Service
RTT	Round Trip Time
URL	Uniform Resource Locator
VM	Virtual Machine

Chapter 1

Introduction

Internet is the fabric for applications and services we use in our daily life such as social networking, news, entertainment and business. Cloud computing has become a way to deploy internet applications without upfront investments in the servers, around the world. Cloud computing refers to computational and application resources provided as utilities over the Internet as "a pay as you go" service [3]. On Infrastructure as a Service(Iaas) clouds, users can create and run virtual machines on demand.

The public cloud infrastructures are available at different locations in the world (Figure 1.1), which enables us to place the applications anywhere in the world. This has many consequences to the internet applications. The on-demand provisioning of resources on cloud enables the application to dynamically adapt its location by creating or moving virtual machines. In addition, the internet application can grow dynamically by adding new virtual machines of its components to satisfy user demands, for scalability and robustness.

Recent literature shows the potential advantage of hosting mobile applications in the cloud to save energy in mobile devices by offloading computation to the cloud [15]. This shows that, in future, more applications will be hosted in the Internet. Unlike the desktop applications, Internet application's Quality of Experience(QoE) is also influenced by the network latency, server location and communication patterns.

Latency introduced due to the distance in network increases the response time of the internet applications. The increasing network traffic, unreliable networks and network congestion add to the communication latency, thereby decreasing QoE of these applications. The best effort delivery model of IP networks provides scalability and flexibility but does not ensure QoE. Even though there are many models to ensure Quality of Service(QoS) for unreliable IP network, limited QoE due to latency of long distance communication remains an issue.

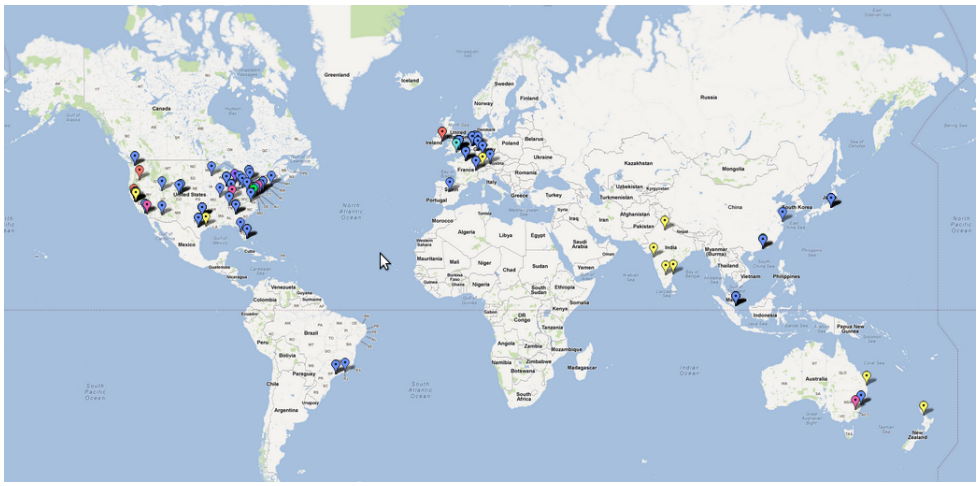


Figure 1.1: Geographical location of different cloud sites. Data was collected from each cloud providers website.

Content Delivery Networks(CDN) were introduced to distribute and cache the contents at the edge of Internet. Distributing contents over the Internet reduces latency to end-users and, core network traffic. CDN, thus improves the performance of content delivery over the Internet. But current Internet Applications do more than static content delivery, whose QoE cannot be ensured by CDNs.

Consider a photo sharing application where you upload photos to the Internet. In order to optimize QoE, the response time must be minimized. If the photo sharing application is hosted in the Internet, the response time for uploading a photo will be in the order of seconds. Uploading a photo to a webserver in user's local network reduces the response time (Figure 1.2). If the photo sharing service can place its server closer to the end-user, it can improve QoE of the end-user. But what if the user is mobile or there are variable number of users from different locations? In such cases, the applications need to adapt to the changing environments to maintain its QoE.

When the internet applications grow dynamically or adapt to the changing environment, by on-demand allocation of virtual machines on distributed public clouds, a strategic placement of application's virtual machines on the cloud would help to improve the QoE. For example, the application creates a new server closer to the user every time the user moves to a new location. This leads to our research question stated in the next section.

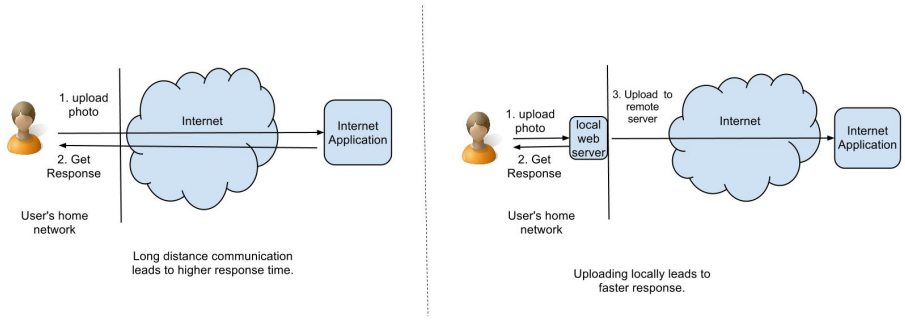


Figure 1.2: The upload process to a local server and server in the Internet

1.1 Research Question

We can dynamically place the application components anywhere on Internet on demand. Can we optimize the QoE of an Internet application by dynamically placing application components closer to end-users?

The research question can be further divided into following sub questions:

- Which factors determine the optimal placement of application components?
- Can we automate the decision making of optimal application component placement?

1.2 Contributions

An application's QoE is affected by its response time. Dynamic placement of application components reduces the response time, though it is not the case for all applications. The improvement in response time depends on the location of all components of the application as well as the communication pattern between them. We introduce a metric called Communication Affinity which quantifies the impact of application component placement on its response time. We propose a method for QoE-optimal placement decision of application components on a distributed cloud, using communication affinity.

1.3 Thesis Outline

To understand the state of the art, we discuss the methods adopted in past to improve QoE and QoS of Internet applications in chapter two. In chapter three we discuss the requirements for the infrastructure that enables dynamic

placement of applications on Internet and identify the basic components of the infrastructure. In addition, we discuss an architecture for the infrastructure.

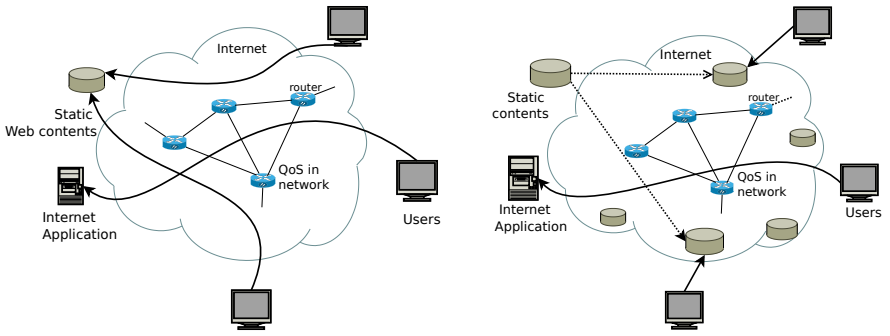
We analyse the impact of the placement of an application components on its response time in chapter four. This study leads to the concept of *Communication Affinity* which is explained in chapter five. Communication Affinity quantifies the effect of latency between various components of an application on its response time. An optimal placement of application components can be determined using Communication Affinity.

The experiments to evaluate the optimal placement strategy using communication affinity is explained in chapter six. The experiments use the architecture and the framework of the infrastructure explained in chapter three for the implementation of dynamic placement of application components. We present the conclusions of our study and the direction for future work in chapter seven.

Chapter 2

Techniques for Improving QoE

Long distance communication over Internet can be inefficient due to unreliable networks, congestion and packetloss. The Internet provided only a point-to point best effort delivery [6]. Integrated Service (IntServ) and Differentiated service (DiffServ) models were introduced to add functionalities to the best-effort IP model to provide QoS for applications [6, 5] (Figure 2.1a). However, end-to-end QoS across multiple domains by IntServ and DiffServ was not employed in Internet. The complexity in policy agreements between multiple providers, for example, agreement for payment and billing limits the adoption of these models by ISPs. Later, programmable networks were introduced to add more application specific processing of the packets [8]. But, none of these models addressed the problem of communication latency, which limits the QoE of the applications.



(a) QoS in network. Communication latency is not solved.

(b) Contents moved into network. Latency between user and content is reduced

Figure 2.1: Evolution of QoE improvement techniques

Other solutions, built on top of the existing internet, were introduced to reduce the communication latency and thereby improving QoE of internet applications. Content Delivery Networks(CDN) [16] improved the efficiency of static content delivery by distributing and caching the contents at different locations in the internet (Figure 2.1b). CDNs were designed for static content delivery; hence it did not solve the problem for other internet applications. The idea of CDN was adopted by edge computing [23] to place applications closer to the users. The edge computing infrastructures are designed for specific type of web applications. Therefore, state of the art methods for improving QoE would not suffice for a wide range of internet applications.

2.1 In-network Solutions

The separation of concerns between the network functionalities and the application was an important design choice in the TCP/IP model which forms the basis of the Internet [10]. In order to provide QoS for applications, new mechanism to provide some control over end-to-end packet delays were added to the basic IP model. These solutions were added to the core of the network which handles packet forwarding and end-to-end packet delivery.

2.1.1 IntServ and DiffServ

Different applications have different QoS requirements. File-transfer, web browsing and e-mail can tolerate delay but they cannot tolerate packet loss, while multi-media applications are latency-sensitive but they can tolerate loss. However, the best-effort IP model provides equal treatment to all packets. In order to facilitate application specific end-to-end QoS in Internet, two models were introduced: Integrated Service model(IntServ) and Differentiated Service model (DiffServ) [6, 5].

IntServ functions on a per-flow basis where QoS is provided for specific packet streams [6]. In order to guarantee the QoS requirements, end hosts specify their requirements using a signalling mechanism. Signalling mechanism reserves resources along the path of the flow (Figure 2.2). End-to-end QoS can be guaranteed only if all network devices in the path support IntServ and agree to reserve resources for the flow. This requires every device in the path to maintain the states which limits its scalability [19].

DiffServ is a coarse-grained mechanism in which each packet is classified into a particular traffic class. Each traffic is identified by a DS codepoint [5]. The routers in the network implement per-hop-behaviours associated with each codepoint, to offer low-loss, low-latency, low-jitter etc (Figure 2.3).

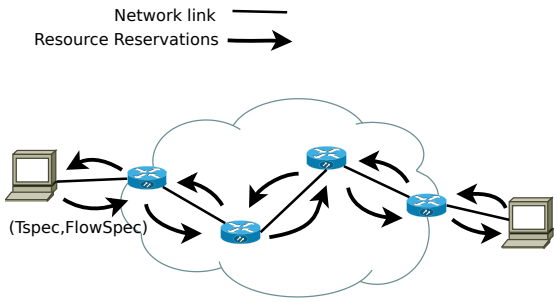


Figure 2.2: Operation of IntServ: Resource Reservation by signalling along the path

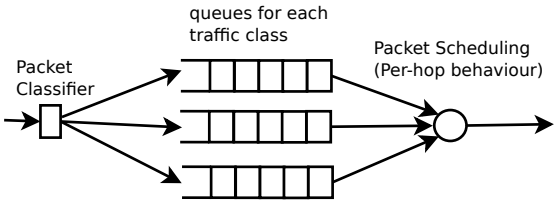


Figure 2.3: Operation of DiffServ at a node: Traffic classification

Both models ensures QoS, how ever the QoE of internet applications depends on the response time, which is limited by communication latency. Ensuring QoS does not reduce the communication latency.

2.1.2 Programmable Networks

Some applications might require additional QoS other than low-delay or low-jitter as provided by IntServ and DiffServ. It may be difficult to include every application specific requirements in network protocols, especially when new applications are emerging [8]. Th programmable network enables applications to program the application specific requirements into the network. Programmable network allows application to change the network, for example, to adapt to a changing environment such as in case of failures.

Active networks [24] enables application specific processing for packets by embedding the processing rules to the packets. Such packets are called capsules. Capsules are executed at each active node along its path (Figure 2.4).

User Programmable Virtualized Networks (UPVN) is a generalized programmable networks model in which applications can control the network elements. In UPVN, programmable network elements are virtualized UPVN [17]. Network elements appear as objects(NC) to the application. NCs can deploy

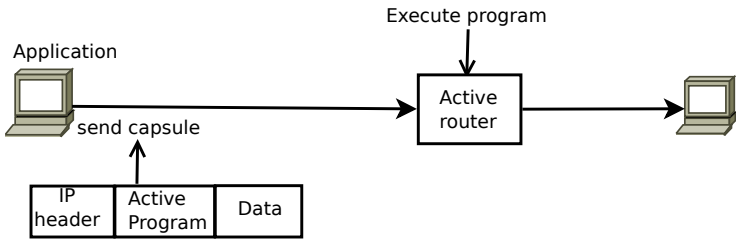


Figure 2.4: Active networks: Active routers process Code embedded in packet.

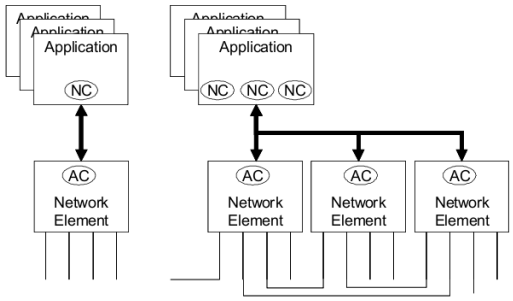


Figure 2.5: UPVN: NE is virtualized in Applicaitons. Application specific code can be run on NE using AC.[17]

application components (AC) on network elements to allow application specific processing (Figure 2.5).

Even though programmable networks enable applications to optimize network for QoS or performance, QoE cannot be ensured.

2.2 Infrastructure based Solutions

QoS models provided in network, IntServ and DiffServ can ensure QoS only if all domains through which the packets travels, supports the QoS model. Moreover, the minimum latency ensured by these models are limited by the distance the packets have to travel. The fact that these models were not deployed in Internet arose the need for other methods to improve QoE of applications. Web content delivery over Internet were not fast due to latency, packet loss etc. Higher the number of hops a packet has to travel, the higher is the chances of packet loss and increased delay. Inorder to minimize the distance to the clients, Content Delivery Networks(CDN) uses additional infrastructures at different parts of the

Internet to host the contents. Edge computing also adopted the idea of CDNs to host web application on additional infrastructures distributed across the internet.

2.2.1 Content Delivery Networks

A Content Delivery Network (CDN) is a distributed system which consists of geographically distributed servers that host content from different content providers [25, 20]. CDNs were designed to reduce the traffic on the network and the latency between content server and the users by distributing and caching the contents on strategically placed servers over the internet. Since the contents are closer to the users, the content transfer is less affected by the latency and packet loss, thus improving the QoE of content delivery.

A CDN consists of an origin server, surrogate servers , request router and a transport system (Figure 2.6). A user request for a content is redirected by a request router to a surrogate server. Request router constantly collects data from surrogate servers and based on the data the user is redirected to one of the Surrogate server. Surrogate server retrieves a copy of the content using the transport system if it does not have a copy of it. The content is cached and then delivered to the user.

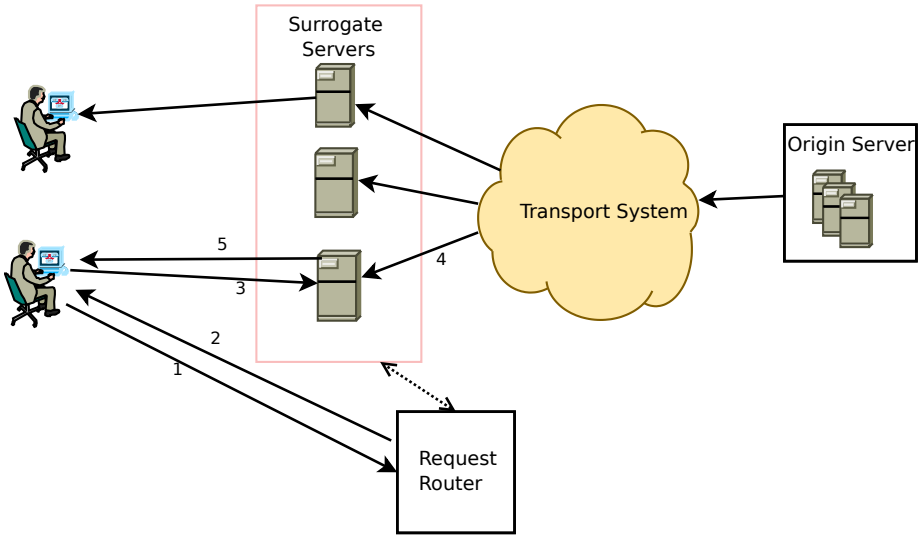


Figure 2.6: Components of a CDN

Origin server: The origin server has the original copy of the contents and maintains the latest updated copy. Origin server can be hosted by the content provider or CDN provider. The content from origin server is propagated to the

surrogate servers by the transport system of CDN.

Surrogate Servers: Surrogate servers replicate or cache contents from origin servers and deliver contents on behalf of the origin servers. Surrogate servers are geographically distributed so as to reduce the network distance to the users. When a request for content is received, surrogate server delivers the content immediately if a valid copy of it is available in the cache. If the content is not cached, it downloads a copy from the origin server using the Transport System and delivers it to the client. A copy of it is cached for future clients. An efficient caching policy must be implemented in order to achieve good cache hit.

Request Router: A Request Router redirects the users to an appropriate surrogate server which is closer to the user. A monitor collects real time measurements on surrogate server load and network connectivity which is used for choosing an optimal surrogate server. The Request Router also performs load balancing by redirecting requests to different surrogate server and thus handle flash crowds and reduce latency perceived by users, if efficiently implemented.

The common methods used for redirecting users are DNS redirection , URL rewriting and http redirection [25, 9]. Http redirection is limited to http services. URL rewriting is useful for partial site content delivery. Here we discuss DNS redirection because it is a common redirection methods used in CDNs and it can be used with any internet application.

DNS redirection: In this method, server's name is mapped to the IP address of appropriate surrogate server when user submits the DNS query [9]. First, user sends the DNS query to the local DNS server. Local DNS server forwards the query to the CDN redirector. CDN redirector would then find the best surrogate server which can serve the content to the user and gives its IP address in return to the query. The advantage of this method is the transparency of redirection. It can be used for any Internet based application. The disadvantage is that it does not consider client IP address when redirecting, because DNS requests may contain the IP address of client's DNS servers. However, the location of client's DNS server might approximates the clients location. Caching of DNS replies in the intermediate DNS servers also limits the ability of CDN redirector to effectively redirect all clients [20].

Transport System: The Transport System distributes content to the surrogate servers. A surrogate server obtains a copy of the content from the origin server or another surrogate server depending on the transport system. The transport system can also be categorize into pull based system and push based system, based on who is responsible for distributing the contents [20].

2.2.2 Edge Computing

Content Delivery Networks makes static content delivery faster by caching it in the network edge. When applications generate dynamic web pages and objects that cannot be cached, CDNs cannot be used to improve the QoE of such applications. Edge computing adopted the idea of CDN, by moving applications to the edge. Application Content Delivery Network(ACDN) and Akamai edge computing are two mechanisms that provide Edge computing facilities.

ACDN (Application content delivery network) is an architecture for replicating the application on edge servers dynamically based on the user demand. If the number of clients from a particular location increases above a threshold, a replica of the application is installed in the server nearest to that location. In [21], a prototype of their mechanism is explained which is tested on a web application which is read-only for users. The dynamic placement decision on ACDN considers only the number of clients at a particular location.

Akamai has an application delivery network and Edge computing platform which is aimed at accelerating application performance by moving computing closer to users. In [18], authors have classified the types of applications they deploy in their edge computing infrastructure. The edge computing platform is limited to a specific type of applications.

ACDN and Akamai support only specific types of web applications. Moreover, the placement decision depends only on number of clients from a location. The users have limited control on the application placement because, they use edge computing provider's private infrastructures.

2.3 Dynamic Placement of Applications

The in-network solutions Intserv, Diffserv and programmable networks are not available in current Internet. Moreover, the QoS provided by these models cannot ensure QoE of internet applications which is limited by communication latency. Even though CDNs improved the QoE of static content delivery, and edge computing improved QoE for specific type of web applications, we still don't have a general method for improving QoE of all distributed internet applications.

CDNs provided efficient content delivery by placing the contents close to where they are needed. In this thesis, we extend the idea of CDNs to the applications. We study whether the dynamic placement of applications close to the users improves the QoE. Dynamic placement of applications is needed to improve QoE in a dynamic environment where clients are mobile, or the number of clients are variable.

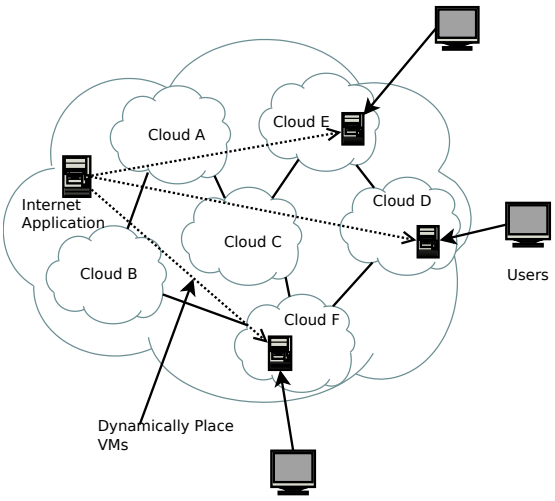


Figure 2.7: Dynamic placement of application on public clouds

Cloud computing provides a platform for users to deploy their applications without the need for buying and maintaining expensive infrastructures. Cloud services are offered at different layers - Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [26]. IaaS cloud offers hardware resources such as computing, storage and networking as services. These clouds are usually hosted in big data centers.

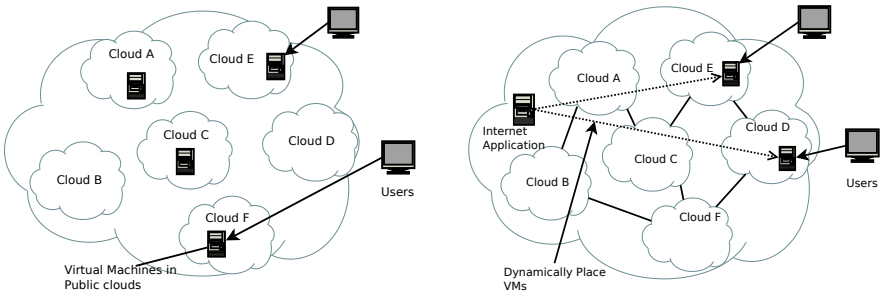
Virtualization is one of the key technology that enables cloud computing [12]. Users can create virtual machines on IaaS clouds on demand where the resources are shared by the users. Virtualization allows easy configuration of user's applications since it abstracts the underlying hardware details. Users can easily manage, create, start or stop his virtual machines on IaaS. These properties of IaaS clouds makes it a good platform for applications that dynamically place its components (figure 3.1b).

We consider a general distributed application decomposed into various components. The components of the applications can be encapsulated in virtual machines and then be placed on the distributed cloud infrastructure on-demand. The QoE improvement method we propose can be used with any distributed internet application.

Chapter 3

Infrastructure Requirements

At present, the applications are statically deployed in the Internet. We can also place our applications on isolated public clouds. Inorder to improve the QoE, the applications can be deployed on the clouds located near to the users. However, static deployment of applications can only provide limited QoE, unless it is over provisioned (figure 3.1a) for unexpected changes in the environment such as network failures and flash crowds. A dynamic placement of application allows to adapt to a changing network environment and changing location of clients. Dynamic placement of application components would provide improved QoE compared to a static placement, by placing the components where it is needed at that time (figure 3.1b). To enable dynamic placement of applications in Internet, the internet infrastructure must provide additional functionalities.



(a) Localization of application in public clouds

(b) Dynamic placement of application on public clouds

3.1 Requirements

In current Internet, when a user request for an application he is redirected to a specific location where the application is available. However, internet applications which dynamically place its computation in Internet might need to create an application instance at a specific location to improve its QoE. The time at which a service must be created at a new location is decided by the application. Figure 3.1 shows the interactions between various entities that enables a client to access an Internet application which can dynamically place its computation. There are four actions shown in the figure. The order of the actions given in the figure may change. The possible orders of actions are:

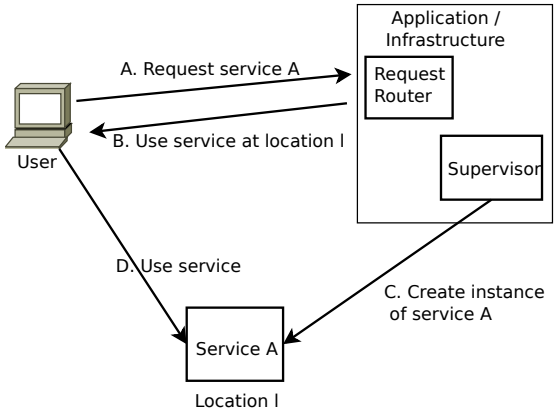


Figure 3.1: Interactions between entities in an application that can dynamically place it services

- A,B,C,D :- When the user requests for a service, user will be redirected to a location. Then the application instantiates a replica of its service at that location which user can access. In this case, the service is instantiated after the user is redirected to the new location. The service instantiation should be done fast because the service should be available when the user requests for it.
- A,C,B,D :- When the user requests for a service, an instance of its service is created at location l. Unlike the previous scenario, the application instantiates the service and the user is redirected when the service is available. This mechanism allows the application to ensure that the service is already running when user requests for it.
- A,B,D,C :- Here the service is instantiated when the request for service comes at location l. Hence, there should be some mechanisms to download, install and run the application when a request for it comes. Service instantiation should be done fast so that user does not notice the delay.

- C,A,B,D :- In this case, the service is already running at location l before the user requests for it. Hence, the user would not notice any delay. This mechanism is useful if the application can predict the possible locations and instantiate the service before requests come.

The basic functions that we require from the infrastructure to enable the above actions in accessing an application are as follows:

- A platform to deploy services dynamically
- A mechanism to choose a location
- A mechanism to transfer service to a specific location
- A request routing service

A platform to deploy services dynamically: The applications need to adapt itself to the changing environment. For example, when a user requests from a particular location, the application creates or move its instance at that location and allows user to access it. This requires an infrastructure that allows dynamic placement of application components. Infrastructure as a Service (IaaS) clouds allow users to create and run their virtual machines on demand on their infrastructure [26]. At present there are many cloud providers located all over the world. Thus it is possible to place our applications any where in the world. Hence IaaS clouds can be used as a platform for dynamic placement of applications.

A mechanism to choose the location: The application decides when to create a VM or move a VM to a new location. Then the application will specify a particular location, a particular node(a machine), or specific requirements of the node to place its vm. When there are many cloud providers and locations, there should be a mechanism for discovering and selecting the locations where the application can place its VMs.

A mechanism to transfer service to a specific location: Dynamic placement of applications in the Internet requires immediate instantiation of virtual machines on specified locations. At present, images of virtual machines created for one cloud site may not be available at another site. Each cloud may not support image formats used in other clouds. Hence, the applications that need to dynamically place VMs on different locations have to create separate VM images for each cloud, which is cumbersome. Hence, the cloud providers must adopt a standard format for virtual machine images.

When an application request to create a VM at a location, the VM image have to be transferred to the specified location. Image transfer might take a long time due to the size of images [22]. Since this is a common problem for all applications, an efficient image management and transfer system must be a part

of the infrastructure to solve this issue. Since CDNs are designed for efficient content delivery, CDN technology may be used for transferring and caching images at different location.

A request routing service: The application dynamically creates and deletes its instances. Hence there should be a request routing mechanism which redirects the user to one of the application instance. The request routing system must know which instances of the applications is running and where they are located. The application may notify the request router about its instances or the request router may implement a monitor which monitors the application.

3.2 Infrastructure

The required components can be organised into a system (Figure 3.2). These components form the basis of the infrastructure that allows dynamic placement of the applications.

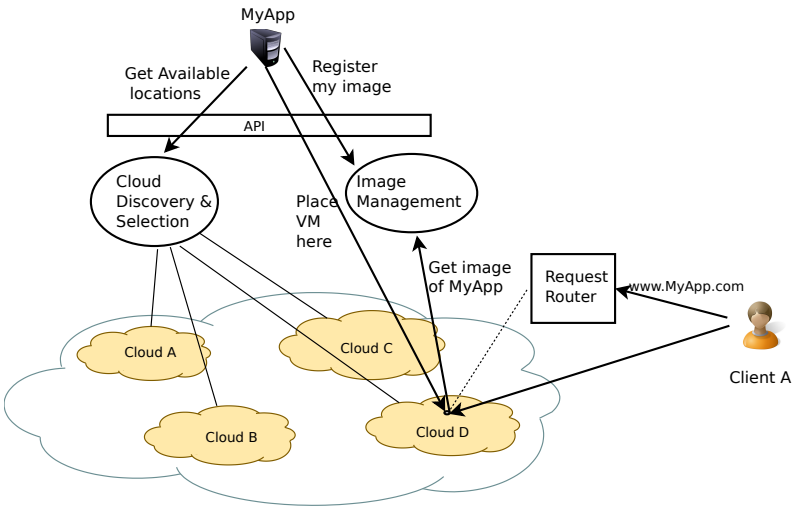


Figure 3.2: Infrastructure that supports dynamic placement of computation

The platform for deployment of applications is a distributed IaaS cloud. Applications choose specific locations to place their virtual machines with the

help of *cloud discovery and selection* component. Application specifies its requirements through an API and retrieves the information about the clouds. For example, if the application need to place the VM at a location near the client, it requests the cloud discovery and selection to get the cloud nearest to the client. In [4] and [7], the authors describe an intercloud reference topology in which this cloud discovery is done by Intercloud Exchange and cloud exchange modules respectively.

Once the application selects a location, it request the infrastructure to place its virtual machine at that location. The cloud at that location has to obtain the image of the application virtual machine to create the virtual machine. This is done through an image management system. *Image management system* provides information about where the images are located and enables the cloud nodes to obtain the image.

The image of the application virtual machine is provided by the application provider. The image would be registered with the image management system and stored in an image repository accessible by all clouds. The images will be identified using a unique id which is used by the application to instantiates its virtual machines.

The request router will be notified when the new application instance is available. The request router redirects the request from the users to the nearest instance of the application.

The optimizer component which makes placement decision is not shown in figure 3.2. We assume that optimizer is a part of the application which interacts with the infrastructure through APIs to dynamically place its components.

3.3 Architecture

A architecture for the infrastructure and the optimizer for dynamical placement of application components is developed based on the study of required components of infrastructure (See figure 3.3). Each component in the infrastructure is mapped to a module in the architecture.

1. Node Discovery: This module implements the cloud discovery and selection. The optimizer communicates with node discovery module to retrieve the information about available nodes where it can place new virtual machines. This information is used by the optimizer to select a suitable node for placing the VM.

2. Request Router: The request router redirects clients to an optimal location where the application is available. The application/optimizer notifies the

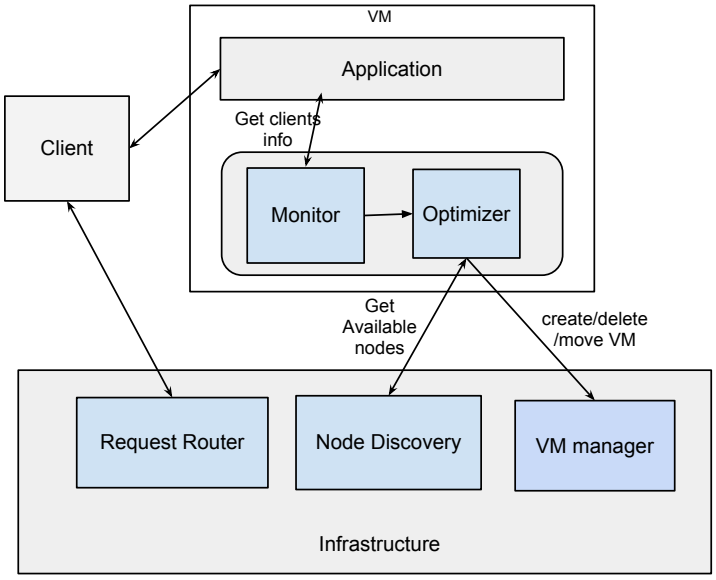


Figure 3.3: Architecture for the infrastructure for dynamic placement of application components.

request router when a new VM of the application is created so that the future clients can be redirected to the newly available VM.

3. VM management: When the optimizer selects a particular node to place its VM, it requests the VM management module to create or move it VM to the new node. VM management module handles the transfer of the image and the creation of the virtual machine.

4. Optimizer: The dynamic placement of application components is handled by an optimizer which interacts with the underlying infrastructure to move virtual machines to specific locations. To enable the application to make the placement decision, the optimizer is made a part of application's VM. Optimizer consists of a monitor which collects information needed for optimizing the location. For example, the monitor collects the details of the client's location.

This architecture is used in the implementation of the experiment environment for simulating the dynamic placement of application for experiments described in chapter six.

3.4 Virtual Infrastructure

Even though the distributed cloud platform is available today, we do not have a complete infrastructure which has all the components we identified. Therefore, we have to simulate the infrastructure for our experiments. Simulating the infrastructure is well suited for our experiments because we could control the factors affecting the experiments in the infrastructure.

We simulated a Wide Area Network on a multi-core machine by introducing latency between the virtual machines. OpenNebula[2] is used for virtual machine management. Each application components were running on virtual machines, which communicate via virtual network bridges (Figure 3.4). Virtual machines are in same network. Latency between the components was emulated by introducing delay to the packets using linux traffic controller - tc [1] and NetEm [13]. (See Appendix D.1 for scripts)

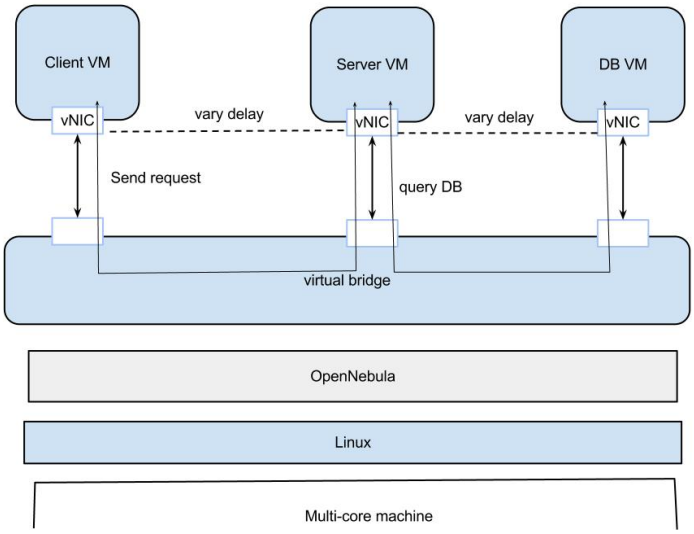


Figure 3.4: Virtual Infrastructure for the experiments

We assume that the virtual machines can be placed on some restricted locations on a 2-D plain. These restricted locations are the available cloud nodes in the infrastructure. The locations on the 2-D plain are represented using (x,y) co-ordinates, where a unit distance in the 2-D plain corresponds to 1ms of latency. A particular point in the 2-D plain is considered as origin. Since the unit distance in the plain corresponds to 1ms of latency, the latency between two virtual machines is calculated by the distance between their locations. A

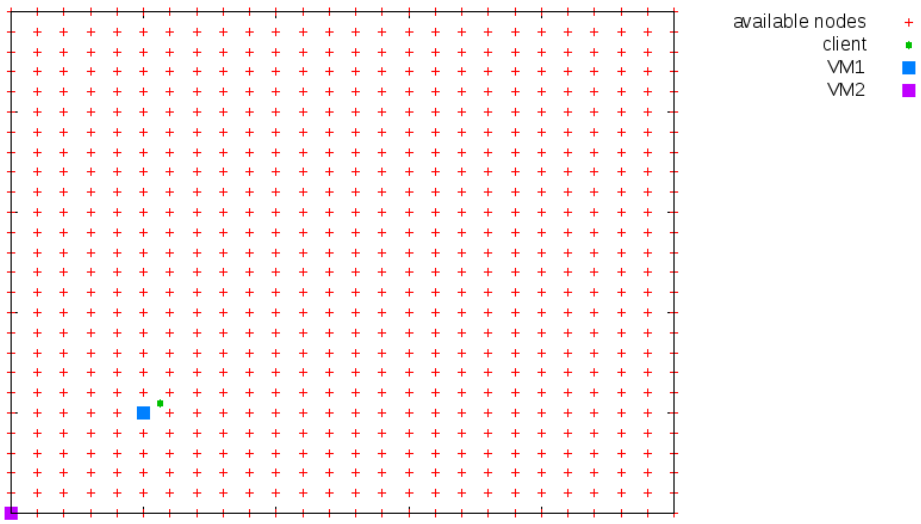


Figure 3.5: The simulation of WAN. The available nodes represents cloud nodes. Possible locations of VMs and clients on the WAN is also shown.

possible locations of clouds and the applications components is shown in figure 3.5.

Movement of virtual machines is simulated by changing the location coordinates of the virtual machine. When a virtual machine is moved, the latency between the virtual machines are changed accordingly.

3.5 Summary

The dynamic placement of the application components is possible using distributed public clouds. The basic components required for the infrastructure was identified. Based on the architecture, a simulated environment is implemented which is used for experiments discussed in chapter six.

Chapter 4

Impact of Application Placement on Response Time

A distributed Internet application can be decomposed into many interacting application components. Moving the application closer to the end-user requires moving all the components, which may not be feasible if there are restrictions to move the components. In order to study the impact of placement of application components on the response time, our experiments place the components of the application on different geographical locations. The interacting application components together determine the resulting response time. The relationship between the latency between application components, and the response time is analysed to decide an optimal placement strategy of the application components.

4.1 Experiment on Amazon Cloud

This experiment is performed to analyse the impact of moving application components closer to the user. The components of the applications are placed at different geographical locations to study how the placement affects the response time.

In this experiment, we considered a photo sharing application $\mathcal{A}$ with a server and a database (See Appendix A for details). The response time is determined by measuring the time to upload a photo to the application and getting the result back (Figure 4.1).

The server and database are placed at different locations in each configura-

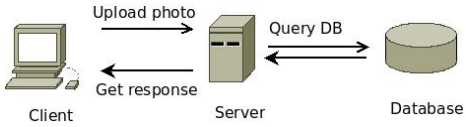


Figure 4.1: Components and interactions in Experiment 4.1

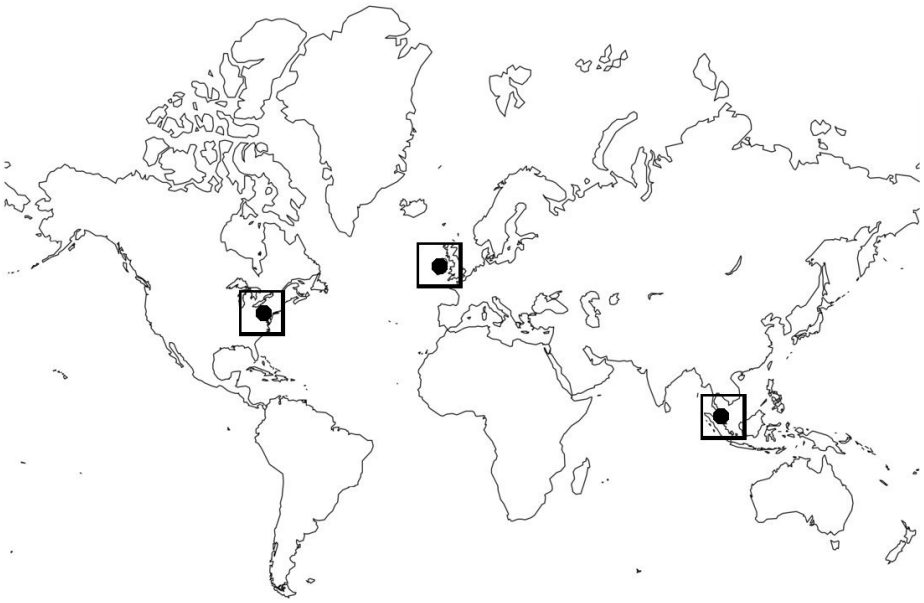


Figure 4.2: Locations of Amazon EC2 used in Experiment 4.1

tion. We use Amazon EC2 cloud at 3 different locations[1] - US(United States), EU(Europe) and SE(Singapore) (Figure 4.2). The table 4.1 shows each configuration and the locations of application and database.

Configuration	Application	Database
Config-1	EU	EU
Config-2	EU	US
Config-3	EU	SE
Config-4	US	US
Config-5	SE	SE

Table 4.1: The location of application and database in each configuration

[1] http://aws.amazon.com/ec2/

The client is located in Amazon EC2 cloud at EU. For each configuration the client uploads photos of different sizes to the application and measures the response time. The photo is uploaded to the application server while some meta data is updated to the database. Figure 4.3 shows the average response time to upload a photo and get the html response for all configurations.

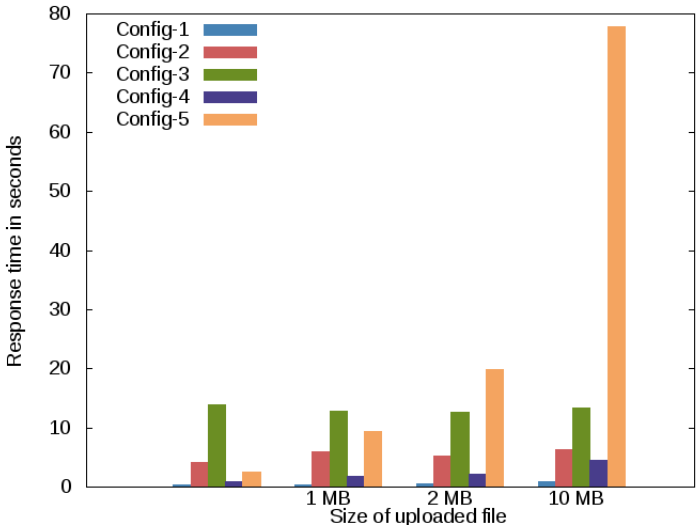


Figure 4.3: Response time of the application for different configurations in Amazon EC2

The experiment shows the effect of communication latency on response time. The best response time is for Config-1 because the client is closest to the application. It is seen that, Config-2 and Config-3 is worse than Config-4 and Config-5, despite the fact that client is closer to the application. This is because for each user request, a series of database queries are issued in order to generate the html response. The latency between the database and the server has a higher impact on the response time than that between the client and the server. Hence, even though the data upload to a nearby application server is faster, the database access increases the overall response time for Config-2 and Config-3.

The overall response time of the application not only depends on the proximity of the client and application but also the location of other components of the application. An optimal placement mechanism for an application should consider all components and their interactions.

4.2 Measuring the impact of latency

The placement of application components affects the response time. We need to study why certain placements configuration in experiment in section 4.1 resulted in large response time. We study how the latency between the components affect the response time, by changing the latency between the components and measuring the response time for different latency. A controlled environment on a multi-core machine to simulate WAN, which is explained in 3.4, is used for the experiment.

In the first experiment, we have a server that runs application $\mathcal{A}$, a database which is used by the server and a client accessing the server. The latency between the client and server is varied between 0 and 200ms. The latency between the server and database was kept at 0ms. The response time for the client request was measured for different latency. Next, the latency between the server and database was changed while latency between the client and server was kept at 0ms. The response time for both cases is measured (Figure 4.4). When the latency between the server and the database is increased, the response time is increasing at a higher rate compared to increasing the latency between the client and the server. This explains the results of experiment in section 4.1. When the server is placed at EU and the database is placed at SE or US, the response time is worse than when the server and the database are kept at same location in SE/US, but far from the client.

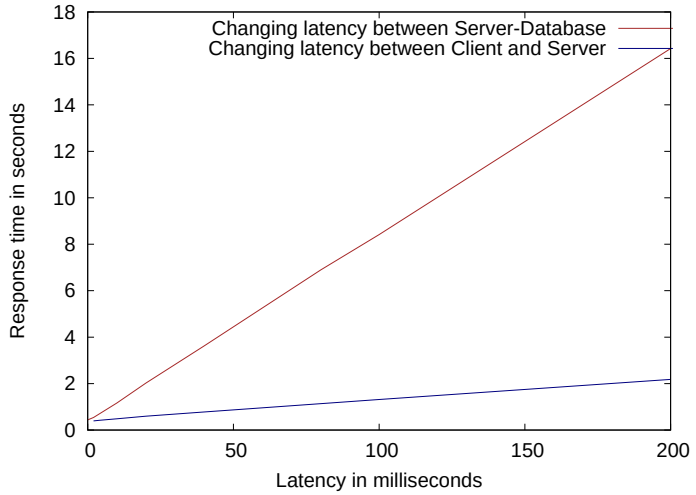


Figure 4.4: Response time of Application $\mathcal{A}$ for different latency between the components.

The experiment was repeated for application $\mathcal{B}$ (Figure 4.5). When the la-

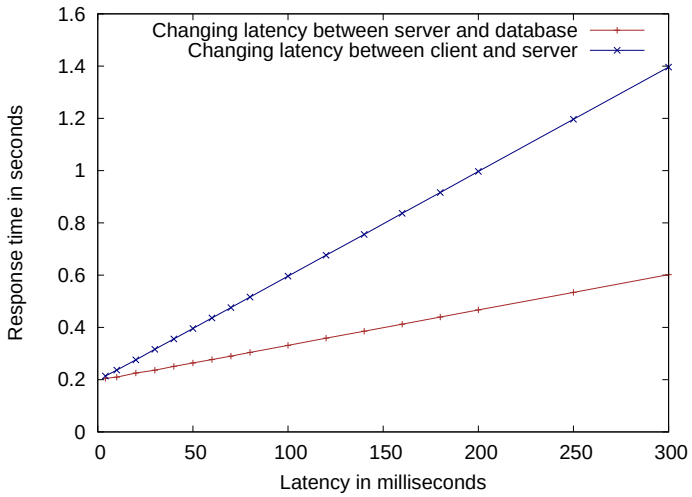


Figure 4.5: Response time of Application $\mathcal{B}$ for different latency between the components.

tency between server and database is increased, the increase in response time is small compared to increasing latency between server and client. This means moving the server closer to the client improve response time.

4.3 Summary

Our experiments show that the latency between different components affects the response time differently for different applications. Response time is not just about latency between the client and the server, but how various components in the application interact with each other. Hence, we need to study the relationship between the latency between the application components and the response time to determine an optimal placement of application components.

Chapter 5

A Framework for Optimal Placement

An optimal placement of applications components depends on the interactions between the components. To minimize the response time, the components has to be placed close to each other to reduce the latency between them. Which components should be placed close together in order to reduce the response time, has to be determined. In hierarchical memory systems, frequently accessed data are placed near the application in a cache. If the data is placed higher up in the hierarchy, the data access is faster; hence the application will be faster. To improve the performance of the applications, the caches exploit the locality of reference of the data [11]. The type of locality is determined by the time and space relationship between the objects.

- Temporal Locality: If an object is accessed at one point of time, then it is likely that same object is accessed again within a short time. Such objects tend to exhibits temporal locality.
- Spatial Locality: If an object is accessed at one point of time, then it is likely that nearby objects are accessed in near future.

Temporal locality of data is exploited to improve performance of memory access in hierarchical memory systems. Most frequently accessed data are kept near to the CPU in cache. Content Delivery Networks also exploit temporal locality of web objects by storing the most frequently accessed data in edge nodes. Ideal case is to cache all objects near the user so that the data access is fast. However, the limited storage in caches forces them to selectively caches objects. Hence the caching algorithms utilize the locality of objects to selectively cache them. It is interesting to find the factor that determines the placement of a component for optimal placement of virtual machines. Can we quantify the temporal and spatial relationship of components? We propose to introduce a term *Communication Affinity* that defines this relationship.

5.1 Communication Affinity

Literally, the term *Affinity* means "a natural liking or attraction to something". Communication affinity is an indication of the coupling between two components of an application. Communication affinity tells how relative placement of two components affects the response time. We define stretch as the ability of two components to be placed far from each other, without affecting the QoE. Two components have high stretch means, these two components can be placed far apart. Communication affinity between two components indicates the stretch of two components.

Definition 1. Communication Affinity between two components is defined as the rate of change of response time with unit change in latency between those two components.

Communication affinity between components A and B denoted by x, indicates that when latency between A and B is increased by 1 unit, the response time increases by x units. Thus communication affinity between two components shows the impact of latency between those two components on response time. Higher the value of x means higher the impact of latency on response time, thus lower the stretch.

Roughly, Communication affinity corresponds to the amount of communication between two components. If the amount of communication between two components is high, then the placement of these two components affects the response time of the application. For example, consider two components A and B that communicates using a sequence of n request/response messages. One request/response message takes one round trip time between A and B. When round trip time (RTT) between A and B is increased by one unit, the total time required for n request/response message increases by n units. If there is another component C which communicates with B using m sequential request/response messages, then increasing RTT between B and C by one unit increases the overall time by m units. Hence the contribution of latency between A and B to the overall time is $n\times$ *latency between A and B* and that between B and C is $m\times$ *latency between B and C*. Here n and m is a measure of amount of communication between components and the communication affinity between them.

We can determine communication affinity by an analysis of the application's communication pattern. However, different communication patterns - synchronous/ asynchronous communication, sequential/parallel communication, request/response messages vs bulk data transfer - behaves differently. A further research in this area would help to analyse different communication patterns and build a model to determine affinity. Here, we determine communication affinity empirically.

The relationship between the latency between the components, and the response time for the applications ($\mathcal{A}$ and $\mathcal{B}$) was found to be linear (Figure 4.4,4.5). Therefore, the communication affinity between the components for these applications is determined by calculating the slope of the straight line got by plotting the response time at different latency between the components. The communication affinities of the two applications are given in table 5.1.

Application	Client-Server	Server-Database
$\mathcal{A}$	8.733	83.33
$\mathcal{B}$	3.995	1.339

Table 5.1: Communication Affinity measured for different applications

In application $\mathcal{A}$, communication affinity between server and database is 83.33. When latency between server and database is increased by 1ms, response time increases by 83.33ms. When latency between the client and the server is increased by 1ms, response time increases by 8.733. Compared to latency between the server and the database, latency between the client and the server has less impact on the response time. On the other hand, in application $\mathcal{B}$, communication affinity between the server and the database is less than affinity between the server and the client. Hence the impact of latency between the client and the server is higher than that between the server and the database.

5.2 Using Communication Affinity for Optimal Placement Decision

The response time at a given latency can be determined using a linear equation with communication affinity, because the latency and response time has a linear relationship. Using this linear relationship we can determine the response time at different placement of application components. We consider a basic internet application which consists of three components - client,server and a database (Figure 5.1).

C_1, C_2 and C_3 are the components of the application.
l_1 is the latency (RTT) between C_1 and C_2.
l_2 is the latency between C_2 and C_3.
A_1 is the affinity between C_1 and C_2
A_2 is the affinity between C_2 and C_3

Let $t_{(m_1,m_2)}$ is the response time of the application when $l_1 = m_1, l_2 = m_2$. $t_{(0,0)}$ denotes the response time of the application when all latencies are 0. Given $t_{(0,0)}$ we can determine the response time at $t_{(m_1,m_2)}$ using Affinity.

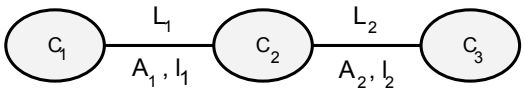


Figure 5.1: A model of application components and the affinity between the components.

$$t_{(m_1,m_2)} = A_1 m_1 + A_2 m_2 + t_{(0,0)} \tag{5.1}$$

Configuration of the application is denoted as $\Omega = (l_1, l_2)$. Let $\Omega_1 = (m_1, m_2)$ and $\Omega_2 = (k_1, k_2)$ be two different configurations of the application. If the current configuration of the system is Ω_1, to determine whether changing to new configuration Ω_2 will improve response time, we can use the following formula.

$$\begin{aligned} \Delta RT &= t_{\Omega_1} - t_{\Omega_2} \\ &= A_1(m_1 - k_1) + A_2(m_2 - k_2) \end{aligned}$$

ΔRT gives improvement in Response Time by moving to the new configuration. Thus we can predict the response time at new configuration and decide whether to change to new configuration or not.

5.3 Optimal placement for application with three components

We determine the optimal placement of a three component application using the communication affinity. The property of affinity such that latency and response time is linearly related simplifies the calculation of the best configuration.

We considered an application with 3 components where C_1 is the client, C_2 is the server and C_3 is the database. The response time of the application is calculated using equation 5.1, for two pairs of values for A_1 and A_2. In the first case, $A_1 = 3.995$ and $A_2 = 1.339$ and in the second case, $A_1 = 1.339$ and $A_2 = 3.995$ The components are assumed to be located in a 2-D plain and the latency between them is calculated as the geometric distance between them. Hence a location in the 2-D plain can be represented as (x,y), where x and y are represented in milliseconds. The database is at origin (0,0) and the client is at (125 ms,125 ms). The response time is calculated for different locations of the server (Figure 5.2). When the communication affinity between the client and the server is higher, the response time is better if the server is kept closer to the client. Otherwise, the response time is better when the server is kept closer

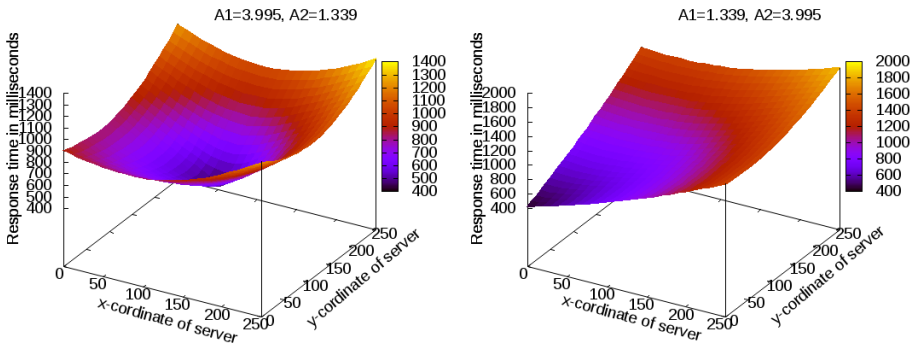


Figure 5.2: Response time calculate for different location of a server. Client is located at position(125 ms,125 ms) and database at origin(0,0). Two plots shows when $A_1 > A_2$ and $A_1 < A_2$

to the database. The optimal placement of an application differs depending on the communication affinity between components.

The response time was calculated for specific latencies other than the location. We consider the case when $A_1 = 3.995$ and $A_2 = 1.339$. Application was configured for different l_1 and l_2 where $l_1, l_2 \in (0, 200)$ and response time was measured for different configuration(Figure 5.3). The location of client is not controllable by the application. The database is also often large that it is costly to move it in terms of time and bandwidth. Hence the restriction to move components(C_1 and C_3)is denoted by keeping $l_1 + l_2 =$ constant. In this case C_2 has to be placed in such a way that RT is minimized. It is seen that given $l_1+l_2 =$ constant, RT is minimum when l_1 is minimum. This is because $A_1 > A_2$.

There is no intermediate location other than a location closest to C_1,for C_2 which reduces the response time. This property can be used in the placement decision algorithm. If $A_1 > A_2$, always find a configuration such that l_1 is minimum. That is always keep C_1 and C_2 closer. The algorithm for finding optimal location of C_2 reduces to finding a location near to C_1.

5.4 Summary

In order to reduce response time we need to find an optimal placement of application components. The ideal case is when latency between all components is 0. In practical applications, there may be restrictions on possible locations where components can be placed. For example, the location of the client cannot be decided by the application. There may not be enough storage to place the

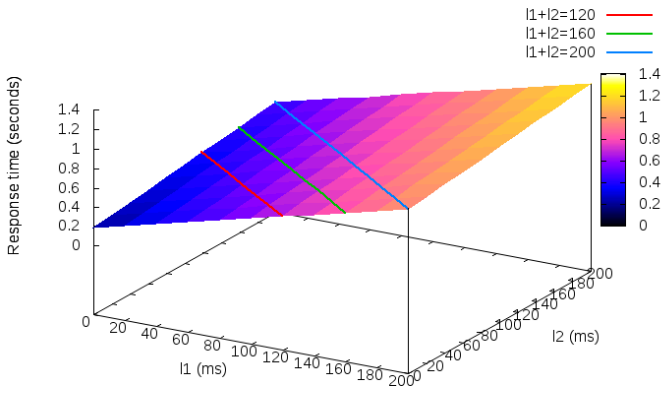


Figure 5.3: Response time for different configurations

database so that latency is 0. In such cases, we may have to place the components on the available locations such that response time is minimized. Given possible placements configurations of an application, communication affinity can be used to decide the optimal configuration which reduces the response time.

Chapter 6

Optimal Placement of Applications

We implement the optimization strategy for applications using the communication affinity and the improvement in QoE is observed. Inorder to observe how communication affinity affects the optimal placement, applications with different communication affinity is considered for the experiments. Using the experiments, we show that the optimal placement of application components is determined by the communication affinity.

6.1 Experiments

The infrastructure for dynamic placement of application is simulated on a multicore machine which is explained in section 3.4. The application components and the clients are hosted on separate virtual machines running on the simulated WAN environment.

The components of the architecture of the infrastructure is simulated. The node discovery module gives the location of all available nodes in the infrastructure where a VM can be placed. The VM manager simulates the movement of virtual machines by changing the (x,y) co-ordinates that represents their location. Four process are running in parallel in the experiment (figure 6.1). The client process is running in client's VM. The optimizer is running in the server. The experiment environment is emulated by two processes - one which simulates movement of clients and the second one which introduces latency between virtual machines.

In our experiments we considered two servers - Server A and Server B. Server A is smart which optimizes its location according to the client's location. It periodically monitors client's location and then moves to an available node

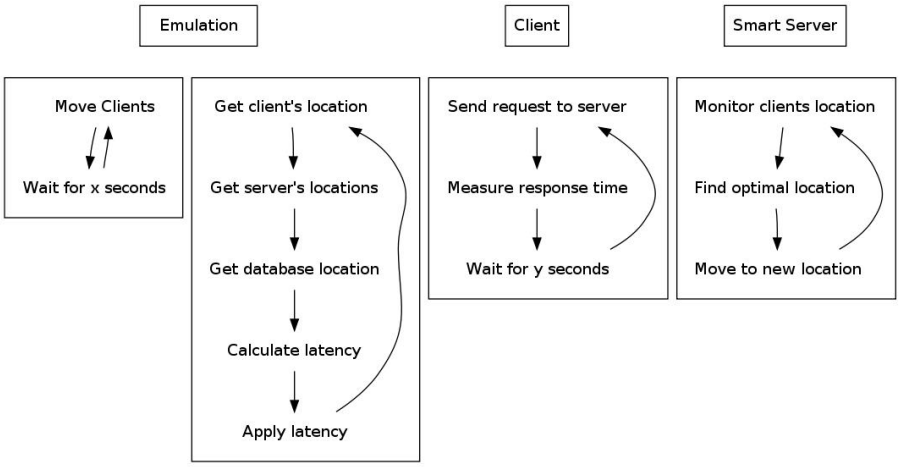
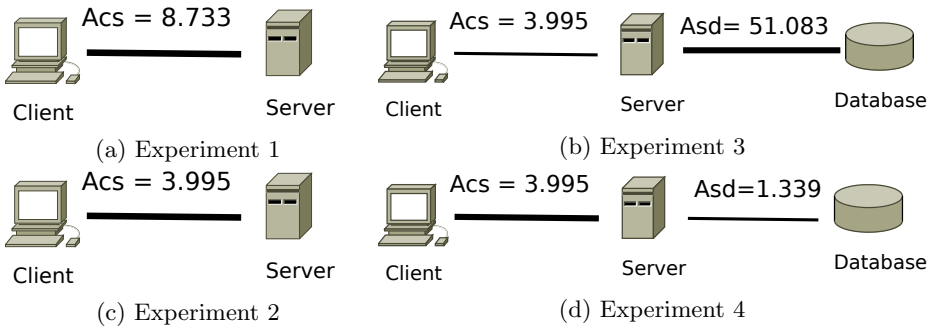


Figure 6.1: Processes in experiments.

nearest to the client. Server B is static which means it does not change its location. The clients are mobile, which move while accessing the servers. The location of static server B is considered as origin. At the start of the experiment Server A is located at origin (0,0). Database is located at (0,10ms). When the experiment starts, the clients periodically increase their co-ordinates by 0.5ms to simulate their moving. All experiments have similar settings except that the application used is different.

The first two experiments consist of applications with only two components - a client and a server with a local database. The two applications has different affinities. In the third and fourth experiments the applications have three components - client,server and database. The ideal case is when both server and database can move closer to the user. Since database is usually large and moving database frequently is costly in terms of time and bandwidth, we consider the case when database is not mobile. Hence the optimization consists of placing the server at an optimal location. In the following experiments we used applications $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ (See Appendix A). The communication affinity between the components of Application A and B is discussed in section 5.1.

Experiment 1. This experiment evaluates whether dynamic optimization of application's location improves the QoE of the application. In this experiment, we use application $\mathcal{A}$. The scenario consists of two clients c_A and c_B that access servers A and B respectively. Server A is a smart server which optimizes its location to improve QoE. Server B is located in a static location. It does not optimize its location. Both clients c_A and c_B are moving while accessing the servers. Client requests to the servers consist of uploading a photo to the application. The measured communication affinity between client and server in



(a) Experiment 1

(b) Experiment 3

(c) Experiment 2

(d) Experiment 4

Figure 6.2: Application settings used in experiments and the measured values of Communication Affinity. (Acs=Affinity between client and server. Asd=Affinity between server and database.)

this application is 8.733 (Figure 6.2a).

Experiment 2. This experiment is similar to experiment 1 with the exception that the application used is application $\mathcal{B}$, which has a different communication affinity value. The application does not allow uploading of photos instead clients can browse through the website. In the experiment client request to view the photo gallery in the website. The measure Communication Affinity between client and server in this experiment is 3.995 (Figure 6.2c). We used an application with different affinity to study how communication affinity affects the QoE.

Experiment 3. In this experiment there is a mobile client, server and database. Server is moving according to clients location, while database is static. The application used is $\mathcal{C}$ whose communication affinity between the components were measured as: $A_1 = 3.995, A_2 = 51.083$ (Figure 6.2b). Here the communication affinity between server and database is higher than Affinity between client and server. This experiment is similar to the experiment in section 4.1. The main difference is that we have used applications with different communication affinities.

The optimal placement strategy using communication affinity tells that the server has to be kept closer to the database because the communication affinity between the server and the database is higher than that between the client and server. In this experiment we have two servers: the static server which is placed close to the database and a moving server which moves with the client. We compare the response time for both servers to find out the optimal placement.

Experiment 4. In experiment 3, communication affinity between the server and the database was large. In this experiment, we used application $\mathcal{B}$ in which communication affinity between the server and the database is less than that

between the client and server. Application $\mathcal{B}$ is similar to $\mathcal{C}$ used in experiment 3, but the database queries are cached to reduce the communication affinity between the server and the database (See Appendix A). The communication affinity between the client and the server is 3.995. The communication affinity between the server and the database is 1.339 (Figure 6.2d). The rest of the experiment is similar to the experiment 3. This experiment would evaluate the impact of communication affinity between the components on its optimal placement.

For this application, the optimal placement strategy is to place the server close to the client, because the affinity between the client and the server is higher than the server and the database. We compare the response time for the static server and moving server to determine the optimal placement.

6.2 Results

Results of Experiment 1 and 2: The response time observed by clients of both servers in experiment 1 is recorded (Figure 6.3). The result shows that response time for the client of smart server is less than that of static server. Hence the response time is minimized by smart placement of application.

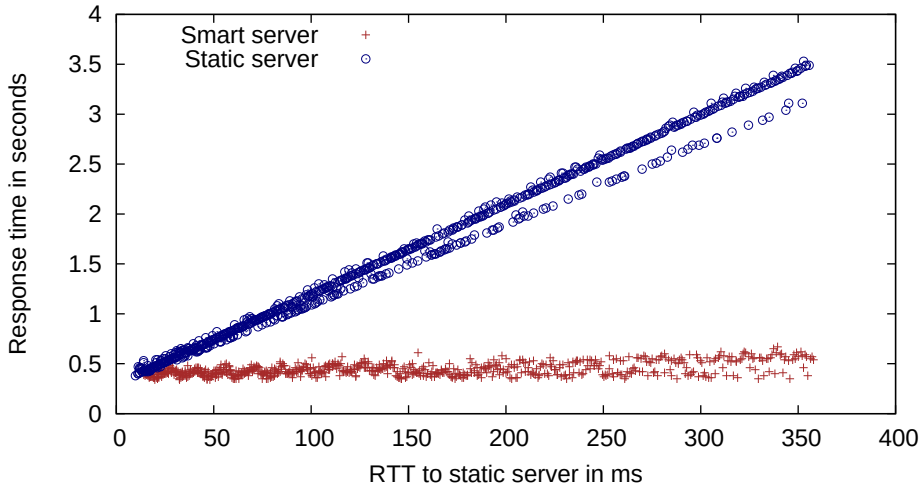


Figure 6.3: Response time for clients accessing Smart server and Static server with application $\mathcal{A}$

In order to compare the effect of affinity on the response time, we compared the improvement in response time for the smart server in both experiment 1 and 2 (Figure 6.4). The decrease in response time for the application with com-

munication affinity 8.733 is higher than that for the application with affinity 3.995. When the communication affinity is high the influence of latency on response time is high. If communication affinity between the components are high, the application will benefit from location optimization. On the other hand, if the affinities are less the application may not have significant improvement by location optimization. Hence finding the affinities between components of an application tells us whether location optimization is required or not.

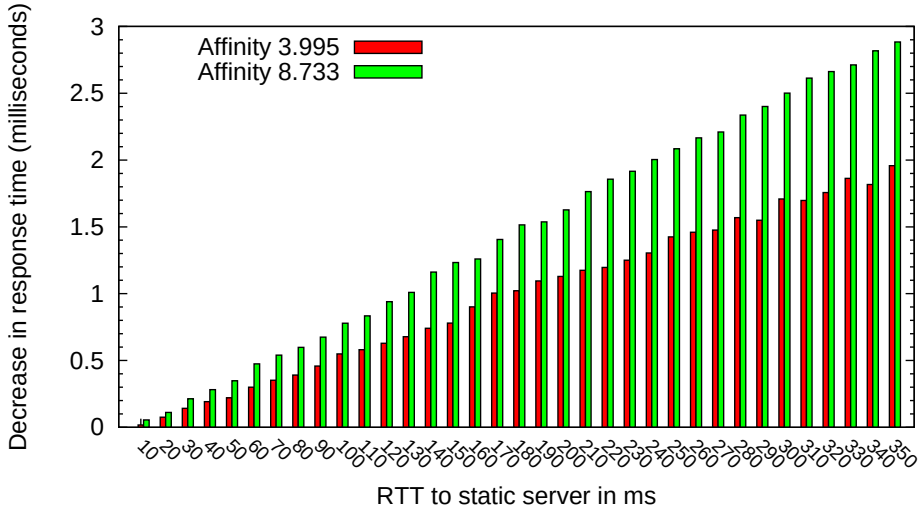
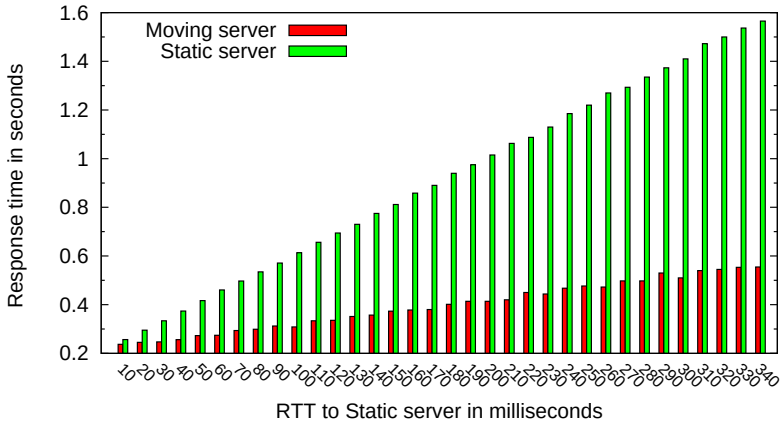


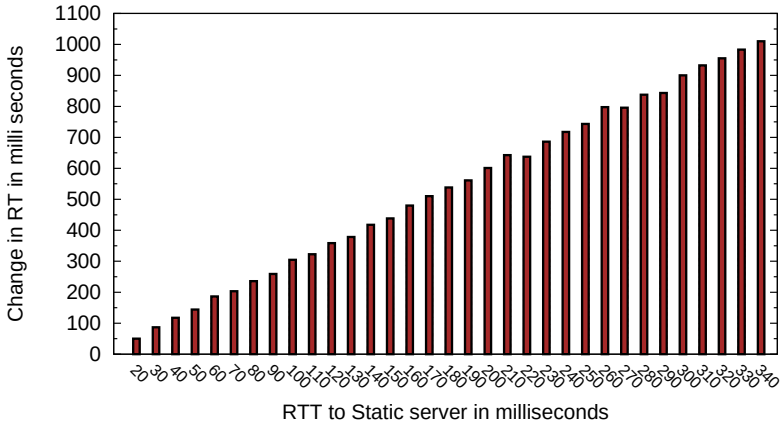
Figure 6.4: Comparing response time for applications with different affinity. Higher affinity application used in experiment 1 and lower affinity application used in experiment 2.

Results of Experiment 3 and 4 : Figure 6.5 shows the response time measured in experiment 4. The response time of smart server is less than that of static server. When the client is moving away, the response time of smart server is also increasing at a small rate. This is because when smart server moves according to the location of client , the distance between the server and the database increases. The latency between server and database adds to the response time. However, the response time is better than the static server. This is because the communication affinity between the client and the server is more than the communication affinity between the server and the database.

The response time for the static server and moving server in experiments 3 and 4 is compared (See figure 6.6). The response time of moving server in experiment 3 is worse than that of the static server. This is because the communication affinity between server and database is higher than that between client and server. The moving server in experiment 4 has better response time than the static server. It is seen that when caching is introduced it increases the



(a) Response time for static and optimizing server



(b) Decrease in response time when server optimizes the location

Figure 6.5: Response time for static and moving server with application $\mathcal{B}$

stretch between the server and the database, since the communication affinity between the server and database is decreased. Thus knowing the communication affinity between various components of an application helps to decide what is the optimal placement of application.

6.3 Multiple Server Placement

This experiment shows an example implementation of optimal placement algorithm for an application with multiple instances of the server. There are also multiple clients accessing each server.

When there are multiple servers and multiple clients, it is not trivial to find

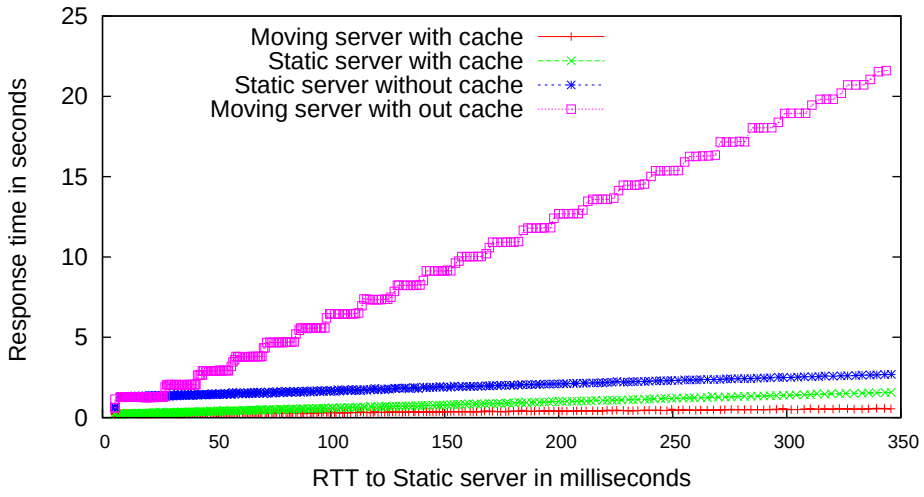


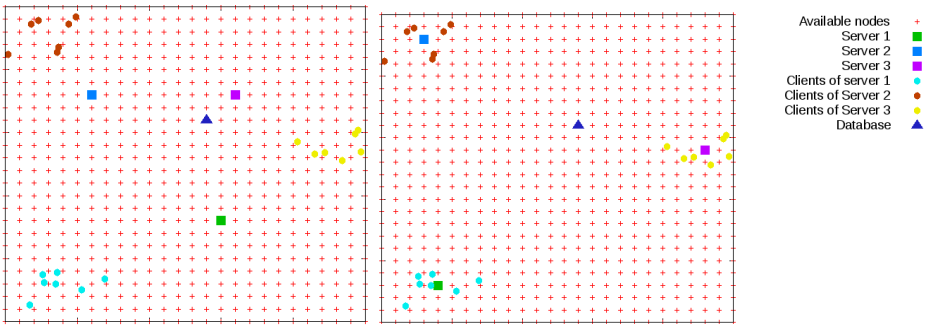
Figure 6.6: Response Time for static and optimizing server 1) when there is no cache 2) when memcache is introduced

an optimal placement of the servers to optimize the QoE. Here we considered an application where communication affinity between the client and the server is higher than the communication affinity between the server and database. Therefore, the optimal placement is to place servers closer to the clients. The application has k instances of servers which can be placed anywhere. The clients requests from different locations, which are redirected by request router to the nearest server.

The goal of the placement algorithm is to assign the clients to different servers and place the servers such that the average distance to their clients is minimum. Instead of average distance, we consider the average of squared distance, which makes the problem simpler. Hence the optimal placement is to find a k-means clustering [14]. We implemented a distributed k-means clustering algorithm. Initially the servers are placed randomly. The request redirector performs the clustering, by redirecting the clients to the nearest server. The server periodically adjusts its location to the mean position of its clients. Each server optimizes its location independent of other servers. After a few rounds, the algorithm will converge and the servers will be in the mean position.

For the experiment we considered the application with 3 servers. The clients are located at different locations. Initially, the servers are located far away from the clients. Figure 6.7 shows the location of servers. After the optimization the servers are placed at a mean position of the clients.

The response time observed by all clients were measured and average re-



(a) Initial locations of servers (b) Location of servers after optimization

Figure 6.7: Locations of servers

sponse time was measured. The experiment was repeated for servers which are static and does not optimize their locations. The average response time for both cases are shown in figure 6.8. The response time for optimal placement is significantly lesser than the random placement.

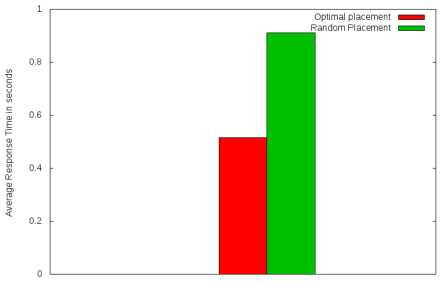


Figure 6.8: Average response time for optimal placement and random placment of server

6.4 Conclusion

The experiments results show that Communication affinity is the metric that determines the optimal placement of application components. Moreover, the communication affinity determines whether the application requires a dynamic optimization of the placement to improve its QoE. If the communication affinity between the components in an application is less, the latency between the components may not have much impact on the response time. Hence calculating communication affinities in a distributed application helps us to categorize applications as those requires optimization and those does not.

Chapter 7

Conclusion

In this thesis, we studied how an optimal placement of application components improves the QoE. In our experiment where we placed application server and database at different location, the results gave an insight that not only the latency between client and application but latency between all components of a distributed Internet application affects the response time.

We introduced the Communication Affinity which models the effect of latency between the components of distributed application on response time. Communication Affinity indicates which components should be placed close to each other in order to minimize response time induced by communication latency.

The results of the experiments where the components of an application move in order to minimize response time, showed that optimizing the location of application components improves QoE. We also showed that if components are placed without considering the communication affinity the response time might not decrease, instead it increases. Hence communication affinity tells us the optimal placement. Thus a placement optimizing algorithm must consider communication affinity to decide an optimal placement.

7.1 Research Questions Revisited

The hypothesis of this research was that *we can improve QoE by dynamically placing application components on Internet*. The experiments show that dynamic placement of application components by considering the communication affinity does improve the response time of application. The questions that were stated related to the hypothesis are the following.

- *What are the factors that determine placement of applications?*

The latency between the the components of an application affects its response time. However, the impact of the latency between the components on response time depends on the amount communication between the components. We quantified it using Communication Affinity. Lesser the communication affinity between two components, more the freedom to place the components anywhere. If communication affinity between two components is high, they have to be placed close to each other.

- *Can we automate the decision making for the optimal virtual machine placement?*
  We showed that response time can be calculated using communication affinity. Hence an algorithm can be developed which can calculate response time at a given configuration and decide an optimal configuration.

7.2 Implications

An internet application which needs to dynamically grow to satisfy the varying user demands, add new servers dynamically to handle the increased number of clients. The dynamic creation of application components is also required to handle the failures. When an internet application wants to create new servers, the optimal placement can be determined using communication affinity.

In a distributed Internet application, components can be located anywhere in the internet. Communication Affinity between the components tells us how much distributed it can be, to ensure a good QoE. In the Internet, where applications can be dynamically placed, an optimal placement that maximizes QoE of all application can be determined using communication affinity. Applications with smaller communication affinity values between the components have more freedom to place its components without affecting its QoE. Hence, the components of applications with small communication affinity may be relocated to enable the applications with large communication affinity between the components to place them where it is needed to improve its QoE.

Moving some of the components of an application might be costly in terms of time, bandwidth or other factors. Communication affinity helps to identify which components have to be moved to improve QoE. This can be used to restrict the movement of some components while maintaining a good QoE and thus reducing the cost involved in moving those components.

7.3 Future Work

In this thesis, we introduced the concept of Communication Affinity which quantifies the effect of latency on response time. The factors such as available bandwidth between the components might also affect the response time. A wider

framework for communication affinity with different parameters should be included in future study. The optimal placement thus should consider not only distance based on latency but also distance based on bandwidth.

A distributed internet application has more number of components and more complex interactions. A component might communicate with two or more components in parallel resulting in a non-linear composition of components. In this thesis we considered a linear architecture. A further research should study how communication affinity framework can be applied to a non-linear component architecture.

We used empirical methods to determine communication affinities in an application. It might be interesting to look into the possibilities of determining communication affinity by methods such code-analysis which can give accurate values which is not affected by environment used for measurements.

In our experiments we considered geographical distance as network latency. In Internet geographical distance may not correspond to network latency. An optimizing algorithm should consider network proximity than geographical proximity. An efficient algorithm which can find an optimal placement algorithm using affinity has to be developed.

The communication affinity framework introduced in this thesis can be a basis for the future work in developing an optimal placement strategy for distributed internet applications which considers various factors that affects QoE.

Bibliography

[1] *http://linux.die.net/man/8/tc.*

[2] Opennebula,http://opennebula.org/.

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

[4] D. Bernstein, D. Vij, and S. Diamond. An intercloud cloud computing economy - technology, governance, and market blueprints. In *SRII Global Conference (SRII), 2011 Annual*, pages 293 –299, 29 2011-april 2 2011.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. IETF RFC 2475.

[6] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. IETF RFC 1633.

[7] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2010*, pages 21–23. Springer.

[8] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, and Daniel Villela. A survey of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 29:7–23, April 1999.

[9] V. Cardellini, M. Colajanni, and P.S. Yu. Dynamic load balancing on web-server systems. *Internet Computing, IEEE*, 3(3):28 –39, may/jun 1999.

[10] Vinton G. Cerf and Robert E. Icahn. A protocol for packet network intercommunication. *SIGCOMM Comput. Commun. Rev.*, 35:71–82, April 2005.

[11] Peter J. Denning. The locality principle. *Commun. ACM*, 48(7):19–24, July 2005.

[12] I. Foster, Yong Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1 –10, nov. 2008.

[13] Stephen Hemminger. Network emulation with netem. Technical report, OSDL, 2005.

[14] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881 –892, jul 2002.

[15] K. Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51 –56, april 2010.

[16] I. Lazar and W. Terrill. Exploring content delivery networking. *IT Professional*, 3(4):47 –49, jul/aug 2001.

[17] Robert J. Meijer, Rudolf J. Strijkers, Leon Gommans, and Cees de Laat. User programmable virtualized networks. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '06, pages 43–, Washington, DC, USA, 2006. IEEE Computer Society.

[18] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44:2–19, August 2010.

[19] Cisco White Paper. Diffserv - the scalable end-to-end quality of service model. August 2005.

[20] Mukaddim Pathan and Rajkumar Buyya. A taxonomy of cdns. In Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali, editors, *Content Delivery Networks*, volume 9 of *Lecture Notes in Electrical Engineering*, pages 33–77. Springer Berlin Heidelberg, 2008. 10.1007/978-3-540-77887-5_2.

[21] Michael Rabinovich, Zhen Xiao, and Amit Aggarwal. Web content caching and distribution. chapter Computing on the edge: a platform for replicating internet applications, pages 57–77. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[22] M. Schmidt, N. Fallenbeck, M. Smith, and B. Freisleben. Efficient distribution of virtual machines for cloud computing. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 567 –574, feb. 2010.

[23] Swaminathan Sivasubramanian, Guillaume Pierre, Maarten van Steen, and Gustavo Alonso. Analysis of caching and replication strategies for web applications. *Internet Computing, IEEE*, 11(1):60 –66, jan.-feb. 2007.

[24] D.L. Tennenhouse and D.J. Wetherall. Towards an active network architecture. In *DARPA Active Networks Conference and Exposition, 2002. Proceedings*, pages 2 –15, 2002.

[25] Athena Vakali and George Pallis. Content delivery networks: Status and trends. *IEEE Internet Computing*, 7:68–74, November 2003.

[26] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.

Appendix A

Applications

All applications are installed in virtual machines with LAMP (Linux, Apache, MySQL, PHP). Linux flavor used is Ubuntu server 10.04.

Application $\mathcal{A}$

This application consists of a php application named *phtagr*[1]. The database is MySQL. The application allows clients to login and upload photos to their account.

Application $\mathcal{B}$

This application consists of php application named *phtagr*. The database is MySQL. Phtagr application is modified to use memcached[2] to cache database query results. The application allows to browse through the photos. It is not used to upload photos. The modifications to the code to enable memcache is given in appendix C.

Application $\mathcal{C}$

This application consists of php application named *phtagr*. The database is MySQL. The application allows to browse through the photos. It is not used to upload photos. Unlike Application $\mathcal{B}$, it does not cache database query results.

[1] http://www.phtagr.org/
[2] http://memcached.org/

Appendix B

Softwares

The versions of softwares used in the experiment. These softwares were installed in the application server.

apache2	2.2.14-5ubuntu8.9
libapache2-mod-php5	5.3.2-1ubuntu4.15
php5	5.3.2-1ubuntu4.15
php5-memcache	3.0.4-2build1
php5-mysql	5.3.2-1ubuntu4.15
libmemcache0	1.4.0.rc2-1
memcached	1.4.2-1ubuntu3
ruby	4.2
ruby1.8	1.8.7.249-2ubuntu0.1
rubygems1.8	1.3.5-1ubuntu2
mysql-common	5.1.61-0ubuntu0.10.04.1
mysql-server-5.1	5.1.61-0ubuntu0.10.04.1
mysql-server-core-5.1	5.1.61-0ubuntu0.10.04.1
php5-mysql	5.3.2-1ubuntu4.15

Appendix C

Application with Memcache

Application $\mathcal{B}$ is a modification of original phtagr application to include caching of queries using memcache. First, memcached has to be enabled in the server. For that install memcached,libmemcache and php5-memcache in you server. Then modify the phtagr source code as described here.

1. Modify the file phtagr/Config/core.php to enable Memcache

```
Configure::write('Cache.check', true);
$engine = 'Memcache';
```

2. Modify the file phtagr/Model/Appmodel.php to add the following in the class AppModel

```
var $cache = true;
function find($type, $params)
{
    if ($this->cache) {
        $tag = isset($this->name) ? '_' . $this->name : 'appmodel';
        $paramsHash = md5(serialize($params));
        $version = (int)Cache::read($tag);
        $fullTag = $tag . '_' . $type . '_' . $paramsHash;
        if ($result = Cache::read($fullTag)) {
            if ($result['version'] == $version)
                return $result['data'];
        }
        $result=array('version' => $version, 'data' => parent::find($type, $params),);
        Cache::write($fullTag, $result);
        Cache::write($tag, $version);
        return $result['data'];
    } else {
        return parent::find($type, $params);
    }
}

private function updateCounter()
{
    if ($this->cache) {
        $tag = isset($this->name) ? '_' . $this->name : 'appmodel';
        Cache::write($tag, 1 + (int)Cache::read($tag));
    }
```

```
    }

    function afterDelete()
    {
        $this->updateCounter();
        parent::afterDelete();
    }

    function afterSave($created)
    {
        $this->updateCounter();
        parent::afterSave($created);
    }
```

3. Modify phtagr/cakephp/lib/Cake/Model/Datasource/Database/Mysql.php and replace the functions describe() and getCharSetName() by the following:

```
public function describe($model) {
        $cache = parent::describe($model);
    //$cache=$this->_cacheDescription($this->fullTableName($model, false));
    if (!empty($cache)) {
                return $cache;
     }
    //$cache = parent::describe($model);
    $cache=$this->_cacheDescription($this->fullTableName($model, false));
    if( $cache != null){
            return $cache;
     }
    $table = $this->fullTableName($model);
    $fields = false;
    $cols = $this->_execute('SHOW FULL COLUMNS FROM ' . $table);
    if (!$cols) {
       throw new CakeException(__d('cake_dev', 'Could not describe table for %s', $table));
    }

     while ($column = $cols->fetch(PDO::FETCH_OBJ)) {
     $fields[$column->Field] = array(
               'type' => $this->column($column->Type),
               'null' => ($column->Null === 'YES' ? true : false),
               'default' => $column->Default,
               'length' => $this->length($column->Type),
     );
     if (!empty($column->Key) && isset($this->index[$column->Key])) {
               $fields[$column->Field]['key'] = $this->index[$column->Key];
     }
     foreach ($this->fieldParameters as $name => $value) {
               if (!empty($column->{$value['column']})) {
                    $fields[$column->Field][$name] = $column->{$value['column']};
               }
     }
     if (isset($fields[$column->Field]['collate'])) {
               $charset = $this->getCharsetName($fields[$column->Field]['collate']);
               if ($charset) {
                    $fields[$column->Field]['charset'] = $charset;
               }
          }
     }
```

```
        $this->_cacheDescription($this->fullTableName($model, false), $fields);
        $cols->closeCursor();

             echo print_r($fields);
        return $fields;
}

public function getCharsetName($name) {
    if ((bool)version_compare($this->getVersion(), "5", ">=")) {
        /*$r = $this->_execute('SELECT CHARACTER_SET_NAME FROM
             INFORMATION_SCHEMA.COLLATIONS WHERE COLLATION_NAME = ?', array($name));
           $cols = $r->fetch(PDO::FETCH_ASSOC);

         if (isset($cols['CHARACTER_SET_NAME'])) {
                  return $cols['CHARACTER_SET_NAME'];
          }*/
   return 'UTF-8'; /*replace this by the character set of the database.*/
   }
    return false;
}
```

Appendix D

Scripts

D.1 Change latency

#Ruby

```
def start(dev)
        system("sudo tc qdisc del dev "+dev+"  root")
        system("sudo tc qdisc add dev "+dev+" root handle 1: prio bands 4")
        system("sudo tc qdisc add dev "+dev+" parent 1:1 netem delay 0ms 0ms ")
        system("sudo tc qdisc add dev "+dev+" parent 1:2 netem delay 0ms 0ms ")
        system("sudo tc qdisc add dev "+dev+" parent 1:3 netem delay 0ms 0ms ")
        system("sudo tc filter add dev "+dev+" protocol ip parent 1:0 prio "+"3"+" u32
             match ip dst "+"0.0.0.0"+"/0  match ip src "+"0.0.0.0"+"/0 flowid 1:"+"3")
end

def stop(dev)
        system("sudo tc qdisc del dev "+dev+"  root")
end

def add(dev,prio,srcip,dstip)
        cmd="sudo tc filter add dev "+dev+" protocol ip parent 1:0 prio "+prio.to_s()+" u32
             match ip dst "+dstip+"/32  match ip src "+srcip+"/32 flowid 1:"+prio.to_s()
        system(cmd)
end

def change(dev,latency,var,prio)
        cmd="sudo tc qdisc replace dev "+dev+" parent 1:"+prio.to_s()+" netem delay "+
             latency.to_s()+"ms "+var.to_s()+"ms "
         system(cmd)
end
```

D.2 Client

The script running in client which sends requests to the server and monitor response time. The following script is for the client in experiment 1 in chapter 6.

#Ruby

```
ipPrefix="172.16.100."
ipStart=1
ipMax=3
url="index.html"

ip=ipPrefix+"1"
changeip="sudo ifconfig eth1 "+ip
system(changeip)

cmd="/usr/bin/time -a -f %e -o"
system("curl --dump-header cookies.txt --form _method=POST
      --form data[User][username]=admin --form data[User][password]
      =adminpas http://172.16.0.3/phtagr/users/login")
i=0
while File.exists? "/home/ubuntu/startprocess"
        ip=ipPrefix+"1"
        changeip="sudo ifconfig eth1 "+ip
        system(changeip)
        puts ip

        system("ping -c 1 -W 1 172.16.0.4" )
        system("ping -c 1 -W 1 172.16.0.4 > pingout" )
        system("awk -f parseping pingout > pingtime")
        system("echo -n `cat pingtime` ' ' >> results3/time"+ip+".out")
        upload=cmd +"results3/time"+ip+".out curl --cookie cookies.txt
                --form _method=POST --form data[File][upload][]=@a.jpg
                http://172.16.0.3/phtagr-localdb-cache/browser/quickupload > out"
        system(upload)

        ip=ipPrefix+"2"
        changeip="sudo ifconfig eth1 "+ip
        system(changeip)

        system("ping -c 1 -W 1 172.16.0.4" )
        system("ping -c 1 -W 1 172.16.0.4 > pingout" )
        system("awk -f parseping pingout > pingtime")
        system("echo -n `cat pingtime` ' ' >> results3/time"+ip+".out")

        upload=cmd +"results3/time"+ip+".out curl --cookie cookies.txt
                --form _method=POST --form data[File][upload][]=@a.jpg
                http://172.16.0.4/phtagr-localdb-cache/browser/quickupload > out"

        system(upload)
        sleep 1

        i=i+1
        if i >= 20
                i=0
                system("curl 172.16.0.3/deletefiles.php")
                system("curl 172.16.0.4/deletefiles.php")
        end
end
system("curl 172.16.0.3/deletefiles.php")
system("curl 172.16.0.4/deletefiles.php")
```

For clients of experiment 2,3 and 4 use the following curl script to browse

through the website

```
curl --cookie cookies.txt  http://host/phtagr/explorer > out"
```

The full scripts used for all experiments are available online at: https://github.com/deepthidev/optimizerscripts