ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE

ENGINEERING AND TECHNOLOGY

RESOURCE MAPPING OPTIMIZATION FOR DISTRIBUTED CLOUD SERVICES

Ph.D. THESIS

Atakan ARAL

Department of Computer Engineering

Computer Engineering Programme

NOVEMBER 2016

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE
ENGINEERING AND TECHNOLOGY

RESOURCE MAPPING OPTIMIZATION FOR DISTRIBUTED CLOUD SERVICES

Ph.D. THESIS

Atakan ARAL
(504122502)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Asst. Prof. Dr. Tolga OVATMAN

NOVEMBER 2016

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DAĞITIK BULUT HİZMETLERİ İÇİN KAYNAK EŞLEMENİN İYİLEŞTİRİLMESİ

DOKTORA TEZİ

Atakan ARAL
(504122502)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Asst. Prof. Dr. Tolga OVATMAN

KASIM 2016

Atakan ARAL, a Ph.D. student of ITU Graduate School of Science Engineering and Technology 504122502 successfully defended the thesis entitled “RESOURCE MAPPING OPTIMIZATION FOR DISTRIBUTED CLOUD SERVICES”, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Dr. Tolga OVATMAN** Istanbul Technical University	....
Jury Members :	**Assoc. Prof. Dr. Feza BUZLUCA** Istanbul Technical University	....
	Prof. Dr. Nadia ERDOĞAN Istanbul Technical University	....
	Assoc. Prof. Dr. Öznur ÖZKASAP Koç University	....
	Prof. Dr. Can ÖZTURAN Boğaziçi University	....

Date of Submission : 20 September 2016
Date of Defense : 3 November 2016

Sabah güneşime,

FOREWORD

First of all, I would like to express my deep appreciation to Dr. Tolga Ovatman, who is my mentor and advisor, and who has created the productive research environment in which I have performed my graduate studies. He has been overly generous with his time and his genius and has provided guidance at key moments in my work while also allowing me to work independently the most of the time. It has been an honour to be his first Ph.D. student.

I would like to thank Dr. Can Özturan and Dr. Feza Buzluca for contributing their time as members of my thesis committee and for their insights and comments which greatly improved this work. I also thank jury members of my thesis defense and anonymous reviewers of the publications who have helped me shaping the final product.

I have collaborated with and learned a lot from many members of the Faculty of Computer and Informatics Engineering. I would like to specifically mention Dr. Berk Canberk for being such a good mentor and Gökhan Seçinti for patiently answering my clueless questions about computer communications.

My research in this thesis has been supported by several research funds and scholarships, including TÜBİTAK 2211 Graduate Scholarship, TBD Leaders of Technology Graduate Scholarship, NETAŞ Ph.D. Incentive Award, and ITU Scientific Research Projects. I would like to thank the funding institutions for their support.

I would like to thank my dear friend and colleague, Doğan Altan, who has provided me a lot of support and encouragement over the years of this thesis. I am also thankful to him for enduring my anxiety attacks and making me smile the whole way through.

My parents have raised me with a love of science and supported my education unconditionally. I have been lucky to have their confidence, patience, pride. My mother have encouraged me through every endeavour with her unconditional love. It was my father who sparkled my interest in computers at my early ages. I am sure he would have been very proud of me, as I am of him.

My loving wife, Nilay, has taken care for everything of my life outside the university, allowing me to freely concentrate on my research. I have spent a lot of time and energy that should have been dedicated to her, but she has never complained. I am in debt to her for her patience and understanding. However, her contribution has not been limited to these. She has listened to every presentation practice of mine and curiously asked me about every research problem that I have encountered. Although she is not a scientist, she has provided me some of the best suggestions and criticism on my work. Without her love and support, I would never be able to conclude this thesis. Thus, I dedicate it to her.

September 2016 Atakan ARAL

TABLE OF CONTENTS

Page

ABBREVIATIONS

SP	: Cloud-based service provider
IP, CP	: Cloud infrastructure provider
NIST	: U.S. National Institute of Standards and Technology
IaaS	: Infrastructure as a service
PaaS	: Platform as a service
SaaS	: Software as a service
VM	: Virtual machine
VN	: Virtual node
WE	: Workflow engine
CDN	: Content delivery network
P2P	: Peer-to-peer
QoS	: Quality of service
FLP	: Facility location problem
MIP	: Mixed integer programming
VNE	: Virtual network embedding
VMCE	: Virtual machine cluster embedding
TBM	: Topology based mapping
PM	: Physical machine
SD	: Standard deviation evenness heuristic
SP	: Span evenness heuristic
RAN	: Random mapping VMCE heuristic
RBN	: Round-robin mapping VMCE heuristic
LUF	: Least-utilized-first mapping VMCE heuristic
LFF	: Least-latency-first mapping VMCE heuristic
cs	: Central data storage
D-ReP	: Decentralized replica placement
KRL	: Known replica locations
LRU	: Least recently used
ppm	: parts-per-million
BCR	: Benefit-cost ratio
PDF	: Probability density function

SYMBOLS

$\boldsymbol{h_i}$	: Demand of customer *i*
$\boldsymbol{d}_{ij}$	: Distance between customer *i* and facility *j*
$\boldsymbol{X}_{ij}$	: Boolean closest facility variable
$\boldsymbol{f_j}$	: Fixed cost of facility *j*
$\boldsymbol{Y_j}$	: Boolean open facility variable
$\boldsymbol{G_C, G_F}$	: Cluster and federation topology graph
$\boldsymbol{V_C, V_F}$	: Vertex sets of the cluster and federation
$\boldsymbol{E_C, E_F}$	: Edge sets of the cluster and federation
$\boldsymbol{A_C^V, E_F^V}$	: Vertex attributes of the cluster and federation
$\boldsymbol{A_C^E, E_F^E}$	: Edge attributes of the cluster and federation
$\boldsymbol{M}$	: Minimum deployment latency
$\boldsymbol{L_i}$	: Latency of the network connection *i*
$\boldsymbol{S}$	: VM size
$\boldsymbol{B}$	: Allocated bandwidth between VMs
$\boldsymbol{D}$	: Size of the data transferred between VMs
$\boldsymbol{\lambda}$	: Level of replica expansion
$\boldsymbol{\alpha}$	: Percentage of expected replica requests
$\boldsymbol{e}$	: Epoch duration

LIST OF TABLES

LIST OF FIGURES

RESOURCE MAPPING OPTIMIZATION FOR DISTRIBUTED CLOUD SERVICES

SUMMARY

The magnitude of data being stored and processed in the Cloud is quickly increasing due to advancements in areas that rely on cloud computing, e.g. Big Data, Internet of Things and mobile code offloading. Concurrently, cloud services are getting more global and geographically distributed. To handle such changes in its usage scenario, the Cloud needs to transform into a completely decentralized, federated and ubiquitous environment similar to the historical transformation of the Internet. Indeed, research ideas for the transformation has already started to emerge including but not limited to Cloud Federations, Multi-Clouds, Fog Computing, Edge Computing, Cloudlets, Nano data centers, etc.

Standardization and resource management come up as the most significant issues for the realization of the distributed cloud paradigm. The focus in this thesis is the latter: efficient management of limited computing and network resources to adapt to the decentralization. Specifically, cloud services that consist of several virtual machines, dedicated network connections and databases are mapped to a multi-provider, geographically distributed and dynamic cloud infrastructure. The objective of the mapping is to improve quality of service in a cost-effective way. To that end; network latency and bandwidth as well as the cost of storage and computation are subjected to a multi-objective optimization.

The first phase of the resource mapping optimization is the topology mapping. In this phase, the virtual machines and network connections (i.e. the virtual cluster) of the cloud service are mapped to the physical cloud infrastructure. The hypothesis is that mapping the virtual cluster to a group of data centers with a similar topology would be the optimal solution.

Replication management is the second phase where the focus is on the data storage. Data objects that constitute the database are replicated and mapped to the storage as a service providers and end devices. The hypothesis for this phase is that an objective function adapted from the facility location problem optimizes the replica placement.

Detailed experiments under real-world as well as synthetic workloads prove that the hypotheses of the both phases are true.

**DAĞITIK BULUT HİZMETLERİ İÇİN
KAYNAK EŞLEMENİN İYİLEŞTİRİLMESİ**

ÖZET

Bulut sistemlerinde saklanan ve işlenen verinin boyutu büyük bir hızla artmaktadır. Bunun başlıca nedeni Büyük veri, Nesnelerin İnternet'i ve mobil kod aktarımı gibi Bulut bilişime dayalı alanlardaki gelişmelerdir. Aynı zamanda, bulut tabanlı hizmetler de hızla daha küresel ve coğrafi olarak dağıtık hale gelmektedir. Kullanım senaryosundaki bu değişiklikler ayak uydurabilmek için Bulut Bilişim geçmişte İnternet'in yaşadığı değişime benzer şekilde tamamen özeksiz, birleştirilmiş ve yaygın bir ortama dönüşmektedir. Bu değişimi sağlayacak araştırma alanları hâlihazırda ortaya çıkmaya başlamıştır. Bulut Federasyonları, Çoklu-Bulutlar, Sis Bilişim, Uç Bilişim, Bulutçuklar ve Nano Veri Merkezleri bunların sadece bir kısmıdır.

Dağıtık Bulut adı verilen bu yeni yapıya geçişte en önemli sorunlar olarak farklı Bulut sağlayıcı ve altyapıları için standartlaşma ve kaynak yönetimi olarak ortaya çıkmaktadır. Bu tezin odak noktası bunlardan ikincisidir: Sınırlı hesaplama ve ağ kaynaklarının özeksizleşmeye uyum sağlamak için verimli şekilde yönetilmesi. Kaynak yönetimi kaynakların belirlenmesi, atanması, izlenmesi ve gelen isteklerin kaynaklarla eşleştirilmesi gibi adımları içermektedir. Bu adımların tümüyle ilgili çalışmalar yapılmış olmakla birlikte asıl katkı kaynak eşleme alanında verilmiştir. Kısaca; sanal makineler, ağ bağlantıları ve veri tabanlarından oluşan Bulut hizmetlerinin çok sağlayıcılı, coğrafi olarak dağıtık ve dinamik bir Bulut altyapısı üzerindeki birimler ile eşleşmesi iyileştirilmiştir. İyileştirmenin amacı hizmet kalitesini düşük maliyet ile artırmak olarak belirlenmiş, ağ gecikmeleri ve bant genişliği ile depolama ve hesaplama maliyetleri çok amaçlı bir iyileştirmeye tabi tutulmuştur.

Kaynak eşlemenin iyileştirilmesi iki aşamalı olarak yapılmaktadır. İlk aşamada sanal makineler ve bunlar arasındaki ağ bağlantıları kaynaklar ile eştirilmekte (topoloji eşleme), ikinci aşamada ise veri tabanındaki nesneler kopyalanıp kaynaklara dağıtılmaktadır (kopyalama yönetimi). İki aşama Bulut hizmetlerinin saklama ve işleme maliyetlerini düşürme ve hizmet kalitesini artırma konusunda birbirini tamamlamaktadır. Hizmet kalitesi artışı; erişim gecikmelerin azaltılması ve hesaplama süresinin kısaltılması gibi iyileştirmelerle mümkün olmaktadır.

Topoloji eşleme aşamasında sanal makineler ve bağımlılıklarından oluşan sanal topoloji, Bulut altyapısı sağlayıcıları ve ağ bağlantılarından oluşan fiziksel topoloji ile eşleştirilmektedir. Bunun için fiziksel topolojinin, sanal topoloji ile izomorfik olan alt çizgeleri belirlenmekte, bunlardan kullanıcıya ortalama gecikmesi en düşük olanı ile eşleme yapılmaktadır. Böylece hem kullanıcıya olan gecikme hem de bağımlı sanal makineler komşu sağlayıcılarda bulunduğundan sanal makineler arasındaki gecikmeler azaltılmış olmaktadır. Ek olarak, bant genişliği de verimli şekilde kullanılmaktadır. Yeterli kaynaklara sahip bir alt çizge bulunmadığı durumda

ise sanal makineleri birbirine olabildiğince yakın sağlayıcılara yerleştiren sezgisel bir yöntem kullanılmaktadır. Ayrıca bir veri merkezi içindeki kaynak atama problemi de ele alınmıştır. Fiziksel makinelerdeki farklı kaynak türlerinin kullanım oranlarının yaklaşık olarak eşit tutulmasının en verimli kaynak kullanımını sağladığı gösterilmiştir.

Bu aşamadaki hipotez, topoloji eşleştirerek yapılan yerleştirmenin, açgözlü algoritmalardan daha iyi sonuçlar vereceğidir. Hipotezin değerlendirilmesi amacıyla yaygın olarak kullanılan Bulut benzetim yazılımı CloudSim geliştirilerek bir deney ortamı kurulmuştur. Yapılan deneyler önerilen algoritmanın diğer yöntemlerde daha düşük gecikmeye yol açtığını göstermektedir. Bu sayede gelen isteklerin yerleşim ve çalışma süreleri kısalmakta ve sistemin toplam iş hacmi artmaktadır. Dolayısıyla, önerilen yöntem hem kullanıcıya hem de sağlayıcıya yarar sağlamaktadır. Üstelik hizmet başına ödenen ücret azalırken, sağlayıcının toplam gelirinde bir değişim olmamaktadır.

Kopyalama yönetimi aşamasının amacı ise veriye erişim süreleri ile veri saklama ve taşıma maliyetlerini düşürmektir. Bulut hizmetlerinin veri tabanları analizlerin ve tutarlılık kontrolünün kolaylığı gibi nedenlerle genellikle merkezi konumlarda tutulmaktadır. Bu nedenle, sanal makinelerin coğrafi olarak farklı konumlarda yürütülüp erişim sürelerinin kısalması sağlansa da, veri ihtiyacı yüksek olan hizmetlerde beklenen yarar görülmeyebilir. Geleneksel olarak veri erişim sürelerinin kısaltılması önbellek kullanımı ile sağlanmaktadır. Ancak dağıtık Bulut sistemlerinde önbellek yönteminin iki önemli sakıncası bulunmaktadır. İlk olarak, çok sayıda ve büyük boyutlu veri nesnelerini tüm uç birimlerde saklamak maliyeti artırmaktadır. İkinci olarak, önbellekteki veri ancak bulunduğu konumda kullanılabilmekte, kullanım oranı düşük olmaktadır. Kopyalama yöntemlerinde ise, her bir kopya çevresindeki birden çok konumdan gelecek isteklere cevap verebildiğinden daha düşük maliyetli bir çözümdür.

Önerilen yöntem, yöneylem araştırma alanından alınmış olan tesis konumu problemini temel almaktadır. Tesis konumu problemi coğrafi olarak dağıtık müşteri taleplerini en düşük maliyetle karşılayacak tesislerin sayısı ve yerlerini belirlemek ile ilgilenir. Kopya yönetimi için uyarlandığında müşteri talepleri veri isteklerine, tesisler ise kopyalara karşılık gelecektir. Böylece hem gecikmeleri hem de maliyeti azaltmayı sağlayan bir amaç fonksiyonu tanımlanmıştır. Her bir depolama birimi yerel bir eniyileyici olarak çalışmakta ve bu amaç fonksiyonunu küçültecek kopyalama, kopya taşıma ve kopya silme kararları almaktadır. İyileştirme algoritması tekrarlı olarak belirli zaman dilimlerinde çalışmaktadır. Böylece kopyalar merkezden uç birimlere doğru adım adım ilerlememektedir.

Önbellek kullanımında önbellekte tutulan verilerin sadece bulundukları birim tarafından bilinmeleri yeterlidir. Ancak, kopyalama yönteminin etkinliği diğer yakın birimlerin de kopyadan haberdar olmasına ve yararlanmasına bağlıdır. Bu nedenle, ikincil katkı olarak, bir kopya bulma yöntemi geliştirilmiştir. Birimlerin tüm Bulut altyapısındaki kopyaların yerlerini bilmeleri önerilen yöntemin özeksiz yapısına aykırıdır. Bu nedenle yeni bir kopya oluşturulduğunda ya da bir kopya silindiğinde sadece gelecekte o veriye istekte bulunacağı tahmin edilen birimlerin haberdar edilmesini sağlayacak bir mesajlaşma yöntemi önerilmiştir.

İkinci aşamadaki hipotez, önerilen kopyalama, kopya yerleştirme ve kopya bulma yönteminin önbellek kullanımına göre hem daha düşük maliyetli hem de daha düşük gecikmeli çözümler üretebileceğidir. Hipotez hem gerçek dünyadan alınmış İnternet erişim izleri hem de yapay olarak üretilmiş veriler ile değerlendirilmiştir. Sonuçlar hipotezi doğrulamaktadır. Ayrıca yapay veriler ile yürütülen deneyler, yöntemin faydasının üretilen istek konumları dağılımının varyansı ile orantılı olarak arttığını göstermektedir. Varyans arttıkça, istekler belirli bölgelerde toplanmakta ve buralara yerleştirilecek kopyaların etkinliği daha yüksek olmaktadır.

Bu tezde, önerilen kaynak eşleme yönteminin her iki aşamasının da başarımının geleneksel yöntemlerden önemli derecede üstün olduğu gösterilmiştir. Dağıtık ve birleştirilmiş Bulut sistemlerinin gerçekleşmesinde kaynakların verimli ve etkin kullanımı önemli rol oynayacaktır. Bu tezde sunulan deneysel sonuçlar dağıtık Bulut sistemlerinin gerçekleşmesi ve geleceği konusunda umut vericidir.

1. INTRODUCTION

Software delivered as services over the Internet, as well as the hardware and software systems that make the delivery possible are referred as cloud computing [6]. In cloud computing paradigm, computational resources such as CPU, memory, bandwidth and storage are treated as utilities that can be scaled up and down on demand. These resources are metered and billed per-usage. Cloud based service providers (SP) do not need to over-provision resources or invest for abundant hardware initially to handle unexpected peak demands. Cloud infrastructure providers (IP), on the other hand, have the opportunity of reallocating idle resources to other clients.

Clouds can be made open to public use by providers (public cloud) or they can be deployed and operated exclusively for an organization (private cloud). Additionally, hybrid clouds support a private cloud but resort to public clouds when it is overutilized. A visualization of the U.S. National Institute of Standards and Technology (NIST) definition for the cloud [7] is presented in Figure 1.1.

Cloud computing services can be categorized into three models according to the level of abstraction they provide. These are, in ascending order of abstraction: (1) Infrastructure as a Service (IaaS) that provide virtual raw resources, (2) Platform as

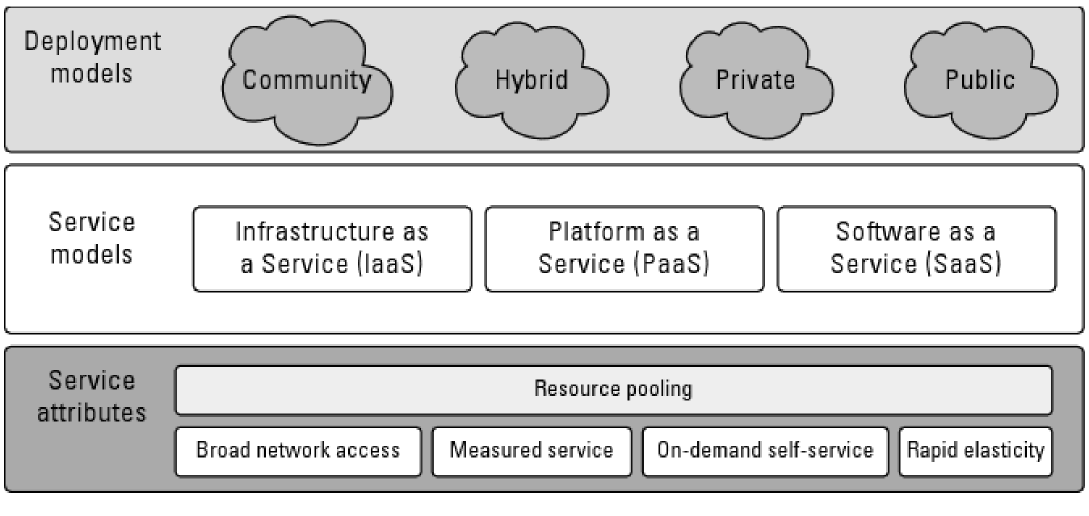


Figure 1.1 : A visualization of NIST cloud definition [3].

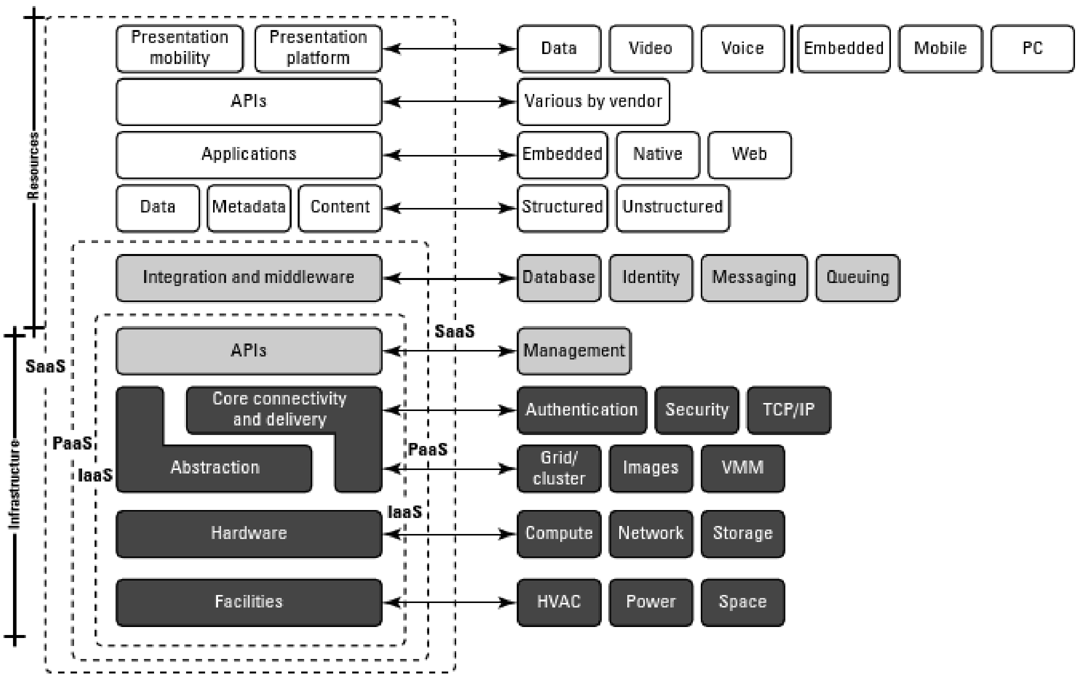


Figure 1.2 : Cloud reference model [3].

a Service (PaaS) that provide virtual development environments, and (3) Software as a Service (SaaS) that provide online applications on demand [8]. Details of these service models are given as reference model in Figure 1.2. As the model demonstrates a cloud based software service (cloud service for short) consists of multiple tiers for presentation, computation, and storage. Presentation and computation tiers are usually deployed as virtual machines (VMs) or containers while the storage tier is usually a database.

1.1 Research Problem

One general research challenge in cloud computing is the efficient mapping of cloud services to the processing and storage resources so that the cloud providers satisfy quality of service (QoS) objectives while minimizing their operational cost [9]. Up to 85 percent of computing capacity remains idle in distributed computing environments [10] and such under-utilization of resources is mainly due to poor optimization of job placement, parallelization and scheduling. The following factors complicate the problem of resource mapping for cloud services.

- The environment is highly dynamic. Resource utilization and prices, user demand, and network conditions vary greatly over time.

- User demand is geographically distributed.
- Cloud infrastructure is geographically distributed.
- There are dependencies and data transfers among the components of the cloud service (VMs and the database).

A smart resource mapping strategy is required to improve QoS by reducing network latency in a cost-efficient way. Here, latency includes; (1) Latency between the user and the cloud service, and (2) Latency between the cloud service components, while the cost includes; (1) VM provisioning cost, (2) Data storage cost, and (3) Data transfer (bandwidth) cost.

In this thesis, network latency is defined and measured between two directly connected computing entities (cloud or edge data centers). It is the duration between a package is sent by on entity and received by the other (one-way) and caused by propagation and transmission delays. The time it takes to prepare the package at the source or to process at the destination is not included. In the case of indirect (multi-hop) connections, individual latency of each constituent are aggregated. Hence, the routing latency is ignored. Given round-trip and processing time, above-defined latency can be estimated as in Equation 1.1.

$$Latency \approx \frac{Roundtrip\ time - Processing\ Time}{2} \tag{1.1}$$

1.2 Hypothesis

Employing resource mapping algorithms that consider and utilize structural characteristics of the distributed cloud services would increase the QoS experienced by service users. Specifically;

1. Mapping the VMs of a cloud service to a subset of data centers that have the similar network topology as these VMs would outperform greedy methods.
2. Mapping the data replicas of a cloud services by considering the level and the location of their demand would outperform data caching.

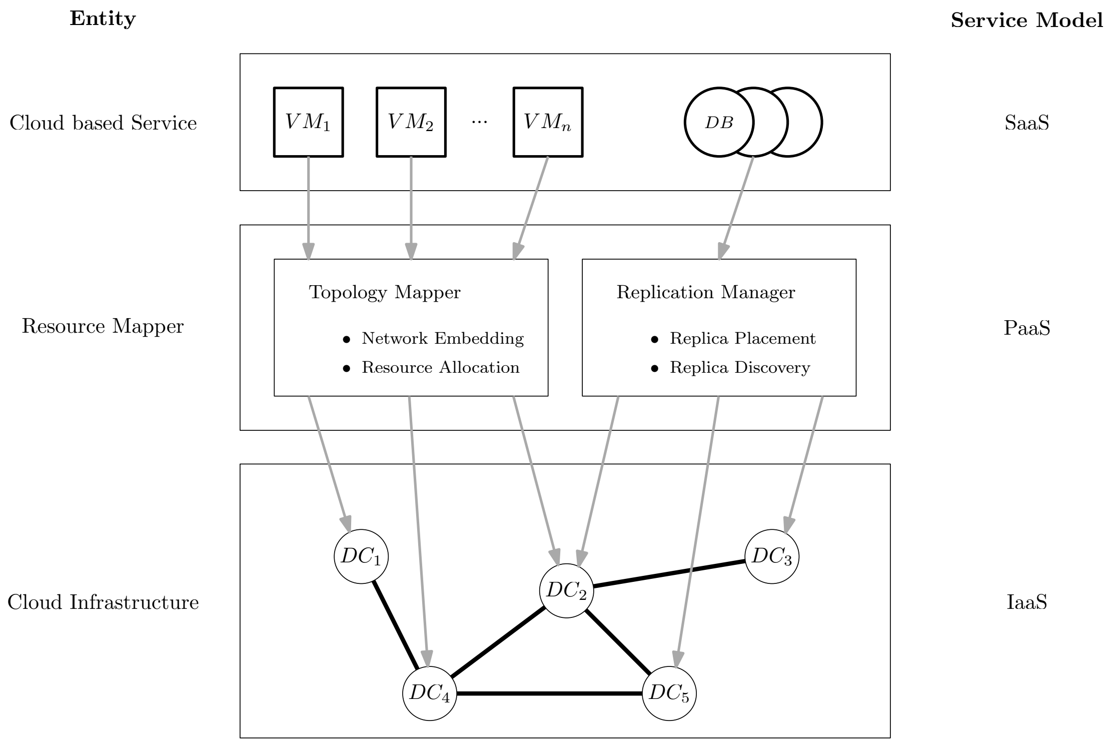


Figure 1.3 : Architecture diagram of the suggested framework.

Such mappings in combination would yield QoS levels that is not possible to achieve with traditional optimization strategies that are not optimized for cloud systems. In addition, they would be more cost-efficient if pricing differences of infrastructure providers, bandwidth and storage space consumption are considered.

1.3 Proposed Solution

A two phase solution for the cloud resource mapping problem is proposed in this thesis. In the first phase (left side of Figure 1.3), VMs and their virtual topology is mapped to the physical topology in an attempt to reduce latency and cost while improving QoS. Both virtual network embedding and resource allocation are carried out in this phase. The second phase (right side of Figure 1.3), on the other hand, optimizes the data access latency and storage cost by means of replica placement and discovery.

These two phases are implemented as the two modules of a resource mapping framework that runs in the PaaS service model as a middleware between the cloud service and infrastructure. The framework is developed and evaluated not to optimize a single service, but the complete demand for the cloud infrastructure collectively. A simulation environment is also implemented as an extension to CloudSim which supports Intercloud systems, virtual topologies and data dependencies. Extended simulation environment is used to evaluate the performance of the suggested framework as well as the baselines in a realistic way.

1.4 Contribution

Contribution in this Thesis can be categorized into four resource mapping algorithms that are intended to work in different phases of cloud resource management as well as the modeling and simulation of the inter-cloud environment. Each of these are summarized in the following subsections.

1.4.1 Topology mapping algorithm

In order to address the network embedding part of the topology mapper phase, a graph theoretical algorithm is suggested. The algorithm conducts an isomorphic subgraph search for the requested VM topology over the substrate inter-cloud topology. When the requested VMs are submitted to the matched nodes of the isomorphic subgraph, the latencies between dependent VMs are minimized. A greedy heuristic approach is executed when the subgraph search is not successful. Proposed algorithm is the first attempt to employ subgraph isomorphism to find a injective match between virtual and physical cloud or grid topologies. Moreover, the study is also the first to explicitly evaluate network latency in the scope of the Virtual Machine Cluster Embedding (VMCE) problem.

1.4.2 Minimum span heuristic

A heuristic approach is proposed for the resource allocation phase. The heuristic makes online decisions to place VMs to physical machines (PMs) efficiently so that more VMs can be accepted before a linear programming solution and accordingly VM migrations are necessary. The main idea is to keep utilization rates of different cloud resources (e.g. CPU, memory, storage, bandwidth) in roughly equal rates so that the prospective exhaustion of one type of resource does not result in wastage of others. Additionally, a mixed integer programming solution that minimizes the number of VM migrations is provided.

1.4.3 Decentralized replica placement algorithm

For the replica management phase of the resource mapping framework, a decentralized replica placement algorithm is proposed. The algorithm uses an objective function derived from the facility location problem to make distributed decisions about migration, duplication and removal of data replicas. It is possible to address the trade-off between latency improvement and cost-efficiency by controling an input parameter. To the best of our knowledge, this is the first completely decentralized replica placement algorithm that work with only local information about the distributed infrastructure.

1.4.4 Replica discovery heuristic

A table based low-overhead replica discovery heuristic is suggested to notify most relevant nodes about nearby replica locations so that they submit their future request to them and receive required data with low latency. Relevant nodes are identified by means of temporal and geographical locality of requests. Distributed nodes keep track of nearby data replicas in their known replica locations tables and update it when they are notified about the creation or removal of a replica.

1.4.5 System modeling and simulation

The inter-cloud environment and the resource mapping problems are represented as a realistic model. Proposed simulator that runs in this model, RalloCloud, is the first simulator for inter-cloud systems.

1.5 Organization of the Thesis

Results of the literature review on topology mapping and replica placement are presented in Chapter 2. Then, in Chapter 3, RalloCloud simulation framework as an extension to CloudSim is proposed. Topology mapper and replica manager modules of the proposed framework are elaborated in Chapters 4 and 5, respectively. Finally, the thesis is concluded with future research directions in Chapter 6.

2. LITERATURE REVIEW

2.1 Preliminary Information

2.1.1 Graph and subgraph equivalence

Subgraph isomorphism and homeomorphism are used in topology mapping (Chapter 4) for finding cloud provider topologies that are similar to the VM topologies so that the VMs are dispatched to the corresponding providers and the inter-latency is optimized. Isomorphism exists between two graphs if a bijective function that pairs vertices of one to the vertices of the other can be defined with edge preserving property. Edge preserving property means that two adjacent vertices of one graph can be paired to two vertices in the other if and only if they are adjacent as well. Instead, if a subdivision of a graph is isomorphic to a subdivision of another graph, then these graphs are called homeomorphic [11]. A subdivision of a graph can be generated by replacing an edge with a new vertex which is adjacent to the endpoint vertices of the original edge with two new edges.

Given two graphs, the objective of the subgraph isomorphism / homeomorphism problems is to find a subgraph of the larger one that is isomorphic / homeomorphic to the smaller. Both these problems are shown to be NP-complete [12] and can only be solved by (1) starting from an empty matching, (2) extending the partial matching by matching a non-matched pattern vertex to a non-matched target vertex, and (3) backtracking if some edges are not matched. Last two steps are repeated until all pattern vertices are matched (success) or all matchings are already explored (failure). Several filtering methods are proposed in the literature in attempt to reduce the search space. In one such filtering methods called the Local All Different (LAD) filtering [13], a set S of pattern/target vertex couples (u,v) to be filtered and a set D_u of possible matches for each vertex of the pattern graph are kept. Initially, all D_u sets contain all vertices from the target graph and S contains all combinations. For each couple (u,v)

Algorithm 1: LAD Filtering [13].

```
Input  : Set $S$ of pattern/target vertex couples $(u,v)$ to be filtered
         Sets $D_u$ of possible matches for $u$
Output: Filtered sets $D_u$ or failure
1 while $S \neq \emptyset$ do
2     Remove a couple of pattern/target vertices $(u,v)$ from $S$
3     if there does not exist a matching of $G_{(u,v)}$ that covers $adj(u)$ then
4         Remove $v$ from $D_u$
5         if $D_u = \emptyset$ then
6             return failure
7         $S \leftarrow S \cup \{(u',v') | u' \in adj(u), v' \in adj(v) \cap D_{u'}\}$
8 return $D_u$
```

from S, a bipartite graph $G_{(u,v)} = (V_{(u,v)}, E_{(u,v)})$ is built with the vertices and edges defined in Equations 2.1 and 2.2.

$$V_{(u,v)} = adj(u) \cup adj(v) \tag{2.1}$$

$$E_{(u,v)} = \{(u',v') \in adj(u) \times adj(v) | v' \in D_{u'}\} \tag{2.2}$$

If there does not exist a matching in this bipartite graph that covers all neighbours of the pattern vertex u ($adj(u)$), then the matching (u,v) is impossible, thus v is removed from the set of possible matches of u (D_u). LAD filtering algorithm is presented in Algorithm 1 as pseudo code [13].

2.1.2 Facility location problem

Facility Location Problem (FLP) is the main idea behind the replica placement algorithm suggested in Chapter 5 to improve cost-efficiency. The problem concerns with the placement of facilities in order to serve the demands of geographically distributed customers with minimum cost [14]. Distance from the customers who will be served by a facility and their demand defines the cost of opening the facility at a certain location. Fixed costs may also be present. Differences in objective functions and constraints lead to several variations of the problem. Here, only the discrete FLP models are discussed for brevity. In such models, the facilities can be opened at finite number of locations so the customer and possible facility locations as well as the distances can be represented with a simple, weighted and undirected graph.

The simplest form of discrete FLP aims to find the location for a single facility which minimizes the sum of distances from all customer locations. Three classes of FLP problems are defined in [15], namely: median, covering and center problems. The number of facilities to be located (k) is foreknown in median problems. Hence, the cost function for each customer is defined as the demand of that customer (h_i) multiplied by its distance to the closest facility location (d_{ij}). Optimum solution is the locations of k factories which minimize the sum of all costs as given in equation 2.3. Here, Boolean variable X_{ij} is set if facility at j is the closest facility to the customer i.

$$Total\ Cost = \sum_i \sum_j h_i \cdot d_{ij} \cdot X_{ij} \tag{2.3}$$

Covering problems introduce the fixed facility opening cost. The objective is to minimize total cost by maintaining a predefined maximum acceptable service distance. The number of facilities does not need to be provided beforehand in this variation. The third class of FLP is referred as center problems in [15]. Here, the goal is to minimize the maximum distance between a customer and the nearest facility given the number of facilities to be located.

All three classes of problems are limited in the sense that they require priori knowledge about either the number of facilities or maximum acceptable distance. A definition of FLP without this limitation is presented in [16]. Equation 2.4 is the cost function using the same notation as equation 2.3 with another Boolean variable Y_j which indicates if a facility is open at j. Moreover, f_j which is the fixed cost of opening a facility at ji is also incorporated. Here d_{ij} should be considered as service cost instead of distance.

$$Total\ Cost \ = \sum_j f_j \cdot Y_j + \sum_i \sum_j h_i \cdot d_{ij} \cdot X_{ij} \tag{2.4}$$

Further variations of FLP include models with limited facility capacities, limited knowledge of parameters (e.g., demands and costs), multiple types of demand, multiple types of facilities, and uncertainty of future parameters [14, 15].

2.1.3 Cloud interoperability

Cloud interoperability which is defined as the ability to dispatch VMs or data between cloud providers, is required to realize distributed and federated cloud infrastructures. It is one of the most important challenges for the adoption of cloud computing [17,18]. Establishing interoperability is also critical for both elimination of vendor lock-in. Several promising approaches for interoperating clouds are already present in the literature [19,20].

Defining open standards is the most straightforward and effective approach for interoperability. A comprehensive list of standardization efforts is presented in [21]. However, a universally accepted standard is not currently available, thus there are also efforts to provide interoperability at the user-level. In Multi-Cloud model, clouds do not directly communicate and the user is responsible for providing an adaptor for their interoperation [22, 23]. A cloud broker who is responsible to negotiate with cloud providers on behalf of the user, may also act as a cloud aggregator and provide a unified interface to nonstandard vendor APIs [24, 25]. In addition to interoperability, cloud brokers can also be employed to ensure QoS with reduced cost in federated cloud [26].

2.1.4 Cloud benchmarking and monitoring

Quality of cloud resource management depends on efficient monitoring and benchmarking of cloud providers. The reason is that the VM scheduling, allocation and mapping algorithms require timely performance and utilization information to cope with quickly changing conditions and to make optimum decisions [27]. Once these measures are collected within a cloud data center, another problem is to disseminate them across cloud brokers in a secure and efficient way so that the providers work in consonance. A large number of studies and tools are introduced which monitor cloud systems [27] and benchmark their services [28] as well as their performance [29].

2.2 Virtual Machine Cluster Embedding

A large number of studies have been conducted in the past years for the efficient embedding of virtual topologies onto federated cloud infrastructures [22, 25, 30–38]. Moreover, similar embedding problems are previously studied in other systems than cloud computing [39–45] or cloud systems without a federation [46–50]. Table 2.1

Table 2.1 : Summary of the literature on virtual machine embedding [1].

	Cloud	Federation	Embedding	Entity	Simultaneous	NW-Aware
TBM	✓	✓	✓	VMs	✓	✓
[30]	✓	✓	✓	VMs		
[31]	✓	✓	✓	VMs	✓	✓
[32]	✓	✓	✓	Services	✓	
[33]	✓	✓	✓	Tasks		
[34]	✓	✓	✓	VMs	✓	✓
[35]	✓	✓	✓	VNs		✓
[22]	✓	✓	✓	VMs	✓	
[36]	✓	✓	✓	VMs		✓
[37]	✓	✓	✓	VNs		✓
[25]	✓	✓	✓	VMs	✓	✓
[38]	✓	✓	✓	VNs	✓	✓
[39]			✓	VNs		
[40]			✓	VNs	✓	✓
[41]			✓	VMs	✓	✓
[42]			✓	VNs		✓
[43]		✓	✓	VNs	✓	✓
[44]		✓	✓	VMs	✓	✓
[45]		✓	✓	VNs	✓	✓
[46]	✓		✓	VMs	✓	✓
[47]	✓		✓	VNs		✓
[48]	✓		✓	Tasks	✓	✓
[49]	✓		✓	VNs		✓
[50]	✓		✓	VNs		✓
[51]	✓	✓		VMs	✓	✓
[52]	✓	✓		Jobs		
[53]	✓	✓		WEs	✓	✓

summarizes some important properties of related studies. Description of each column of the table is given below.

Cloud Whether the placement is made onto a cloud infrastructure with on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service according to the NIST definition [54].

Federation Whether the VMs or tasks are placed onto a federated, distributed and networked infrastructure (e.g. Federated cloud, Multi-cloud, Inter-cloud, etc.).

Embedding Whether a network or VM cluster embedding is carried out such that two graphs are matched. Otherwise, VMs or tasks are individually treated and placed.

Simultaneous Whether the algorithm maps nodes and links simultaneously. Otherwise, there are two independent and/or sequential mapping phases of the the algorithm.

Network-aware Whether the algorithm considers network factor during the placement. This includes the improvement of latency, bandwidth utilization, hop count, etc.

Entity The term used in the study to indicate entities that are placed (Virtual Machines (VMs), Virtual Nodes (VNs), Services, Tasks, Jobs or Workflow Engines (WEs)).

Comments on a subset of the studies in Table 2.1 are provided for the sake of brevity. Authors of the paper [39] propose a mixed integer programming (MIP) solution with relaxed integer constraints for the network embedding problem. The approach increases the coordination between node mapping and link mapping phases with the aim of increasing acceptance rate and revenue.

The first use of subgraph isomorphism for embedding virtual topologies is in [40]. Later, [41] suggested mapping only the critical nodes via subgraph isomorphism and the rest with heuristics to reduce complexity.

Studies introduced thus far are designed to embed virtual networks onto single-site (local) substrate networks such as a data center. In that scenario, network latency is usually low and bandwidth capacity is abundant. Thus, network factors are not as critical as geographically distributed environments, e.g. federated cloud. Additionally, geographical location of the user base emerges as an important factor for the performance of clusters in the latter case.

In the other works that are discussed henceforth however, authors consider a similar distributed resource allocation problem to the one in this thesis. An exact (integer programing) solution to the network embedding problem is suggested in [43]. They also prove that grouping incoming requests during a period and embedding them in batches yield higher acceptance ratio and lower cost in comparison to the individual handling of each request. Authors of the article [25] employ cloud brokers both to optimize placement and to act as a uniform interface for developers. The method for optimized placement is still integer programming. Even though integer programming can produce optimal results in a centralized case, using a heuristic can reduce the amount of overhead induced by solving large optimization problems as well as allowing distributed solutions.

Another study [31] consider both data center selection and VM placement to the physical resources of the selected data centers. Suggested 2-approximation algorithm first finds a subgraph of the complete data center topology with the smallest diameter. Then, VMs requested by the user are partitioned in such a way that each partition fits to a data center in the subgraph and the inter-data center traffic is minimal.

Virtual topology graph is partitioned into connected components via a k-cut algorithm in [38]. Then, a new graph, the nodes of which represent the connected components and the edges of which aggregate the inter-component links, is generated. Later, the generated graph is matched to an isomorphic subgraph of the physical topology and the VMs in each partition is submitted to the cloud matched to that component. In this study, mapping between the VMs of a cluster and clouds is not injective (one-to-one). Since the focus is to balance load and reduce costs, several or even all VMs can be mapped to the same cloud without partitioning if the cloud has enough idle capacity. Hence the mapping would fail to gain the benefits of inter-cloud deployment. Similarly, an iterated local search based graph partitioning and integer programming based embedding is proposed in [35]. The objective function of the embedding minimizes the cost and number of hops. Recently, the same authors suggested a semantic based greedy node mapping algorithm for federated virtual infrastructures [37]. The algorithm defines a upper limit for the number of hops between nodes to increase QoS. However, actual latency incurred by the distribution are not evaluated in neither of these studies. Other approximation techniques to solve the network embedding or similar problems include artificial immune system [42] and Markov random walk [47].

2.3 Local Virtual Machine Mapping

Selection of physical machines in a single cloud data center to host virtual machines while fulfilling the requirements and optimizing resource usage is a problem that can be solved via optimization and approximation algorithms. Some examples include the use of heuristics, linear programming, artificial intelligence, nature inspired computing and game theory [55].

In [56], authors introduce the skewness metric to measure unevenness in the utilization of various resources within a server. They aim to find a trade-off between overload avoidance and green computing concepts. Using a set of threshold based heuristics and a prediction algorithm they dynamically create a list of migrations that relieves overloaded servers to ensure QoS and evacuates underloaded ones to exploit green computing.

Following the technological improvements in the cloud systems, VM migration is being used to increase utilization of the virtual machines on the cloud infrastructure. [57] considers the different VM migration strategies adopted by the host machines that have different load states by taking into account four different resource types. In a more recent study [58], migration is performed regarding the queue model of the time of application deployment requests from the clients. A centralized control management mechanism has been defined to inform the client which server is available at a time.

2.4 Data Replica Mapping

Tables 2.2 and 2.3 summarize the literature on data replica placement. Among these, comments are provided only on the most relevant approaches to the one in this thesis for brevity. The studies are categorized on the basis of three binary classification rules (i.e. Centralized/Decentralized, Complete/Partial Information, and Static/Dynamic). In addition, information on their intended environment, network topology restriction and optimization objectives are provided. More details on the columns of the tables are given below.

Decentralized A check mark (✓) indicates that the replica placement algorithm is executed on multiple locations in parallel. Others determine placements centrally from a single node.

Partial Information Centralized approaches always use complete demand and topological information for the placement. However, some decentralized ones require only local and / or partial information.

Dynamic Replication is dynamic if the number and location of the replicas and or the network status change over time based on the experienced demand and/or cost. In static replication, replicas are not migrated or deleted after their initial creation.

Environment Intended running environment of the algorithm can be one of the following: Cloud, Content Delivery Network (CDN), Cloud-Based Content Delivery Network (Cloud-CDN), Peer-to-Peer System (P2P), Data Grid, Web (including Internet Services), or unrestricted/unspecified (N/A).

Topology If the network topology graph among the nodes is restricted for an approach to work, that is indicated in this column. Alternatives are tree, complete graph, multi-tier, and unrestricted (i.e. any graph).

Objectives This column lists the criteria which are aimed to be optimized via replication. The following objectives are encountered in the literature review.

Proximity indicates the user access time to the replica. Optimizations of network latency, response time, hop count, and distance are grouped under this criterion.

Cost indicates the monetary cost of storing replicas. Methods that do not explicitly consider monetary cost but aims to reduce/minimize the number of replicas are also included.

Bandwidth indicates the network overhead and bandwidth utilization.

Availability indicates the fault tolerance and reliability of the system.

Load Balance indicates the avoidance of hotspots by spreading demand across several nodes.

Another classifications of replica placement algorithms can be found in [59] and [60]. There also exists surveys on dynamic replica placement in data grids [61, 62] for the interested reader.

Centralized methods lack scalability by definition and create a performance bottleneck in replica placement. They are particularly infeasible for large-scale distributed topologies such as the Internet and Cloud. However, the most of the literature in replica placement is centralized approaches due to the extra complexity decentralization brings. First, efficient synchronization of inputs (e.g. demands, latencies, costs) and outputs (e.g. number and location of replicas) of the replication algorithm is challenging especially in a dynamic environment. Second, local knowledge about the environment obstruct optimality of the placement in comparison to global knowledge.

Thus, even some of the decentralized approaches assume global knowledge at each location which causes scalability issues similar to centralized ones.

2.4.1 Centralized methods

Geographic placement of shared data of cloud services is investigated in [63]. Suggested technique places data to the weighted geographical center of their consumers and then maps data to the closest data center by also considering load balance. In [64], first the number of replicas is calculated based on the popularity, recentness and customer-assigned importance of data, and later these replicas are placed by minimizing a distance metric.

In [65], authors suggest a methodology to place data replicas to achieve dual objectives, i.e. to increase security by placing complementary pieces to nonadjacent locations and to reduce data access time by placing them centrally. They employ betweenness, closeness and eccentricity centrality measures as well as graph coloring to achieve these objectives. The algorithm always create a single replica of each data piece irrespective of its popularity. The replica is placed to the node with the maximum dependency to that data.

In addition to replica selection and placement, a request redirection strategy is also proposed in [66]. The algorithm optimizes the cost of data storage and transfer for distributing content to users over storage clouds. A MIP formulation and multiple heuristic solutions are provided. Experimental comparison demonstrate the superiority of online and dynamic heuristics in terms of cost and number of QoS violations.

A two-phase mapping of tasks to replicas and replicas to data centers is suggested in [67]. Proposed genetic algorithm solution assumes that the number of replicas is foreknown and aims to reduce cost and latency by decreasing the number and size of data movements between data centers.

There also exists studies that depend on a specific type of graph topology on which the replicas are distributed. Tree networks are considered in [68] with a specific emphasize on read and write costs. Another tree topology based solution [69] aims to minimize the number of QoS violations in terms of latency. Proposed algorithm exploits already employed replication practice, which is intended for availability, by placing the data of

Table 2.2 : Summary of the literature on centralized replica placement [2].

	Decentralized	Partial Information	Dynamic	Environment	Topology	Objectives
[68]				Web	Tree	Proximity
[72]				CDN	Unrestricted	Proximity
[73]				N/A	Unrestricted	Proximity
[74]				Data Grid	Tree	Proximity, Cost, Load Balance
[75]				N/A	Tree	Proximity
[76]				N/A	Unrestricted	Proximity
[65]				Cloud	Unrestricted	Proximity
[77]				Cloud	Unrestricted	Bandwidth, Load Balance
[67]				Cloud	Unrestricted	Proximity, Cost
[78]			✓	N/A	Unrestricted	Availability
[79]			✓	Data Grid	Multi-Tier	Proximity, Cost, Bandwidth
[80]			✓	CDN	Unrestricted	Proximity, Cost
[63]			✓	Cloud	Unrestricted	Prox., Bandwidth, Load Balance
[81]			✓	Data Grid	Multi-Tier	Proximity
[64]			✓	Data Grid	Unrestricted	Prox., Bandwidth, Availability
[66]			✓	Cloud-CDN	Unrestricted	Proximity, Cost
[82]			✓	Data Grid	Tree	Prox., Bandwidth, Availability
[83]			✓	Data Grid	Multi-Tier	Proximity, Bandwidth
[69]			✓	Cloud	Tree	Proximity, Availability
[84]			✓	Cloud	Unrestricted	Bandwidth, Load Balance
[85]			✓	Cloud-CDN	Unrestricted	Cost, Availability
[86]			✓	Cloud (Mobile)	Unrestricted	Proximity, Availability

applications with high QoS requirements on high-performance nodes. More recently, the combination of replication and erasure coding mechanisms is suggested to leverage availability in Multi-Clouds [70, 71].

Typically, centralized methods yield optimal or near-optimal placement of replicas by making use of medians or centrality metrics. However, such methods have the following drawbacks in comparison to distributed and decentralized methods [87].

- Collecting and transferring the complete system status (e.g. demand for each file, storage cost, network information, etc.) causes a network overhead especially in dynamic and large-scale systems.
- Similarly, distributing control data (e.g. replicated files and their locations) uses up bandwidth and causes delay. This increases the response time of the algorithm.
- Optimization is computationally expensive and does not scale well with the number of nodes.
- Algorithms are usually not iterative. Thus, the complete optimization must be carried out from scratch even for minor changes in the system status.
- The central replica controller is a single point of failure.
- In the case of median-based algorithms, replica count should be given a priori.

2.4.2 Decentralized methods

One of the earliest attempts to dynamic replica placement is [88]. Authors propose a dissemination tree to replicate and synchronize data. The aim is to place minimum number of replicas on data access paths while respecting latency guarantees and balancing load. In addition to replication, caching is also employed to that end.

A file replication algorithm which places the replicas on the so called traffic hubs on the data access paths in a P2P system is proposed [89]. Assumption is that, proposed replica placement will be more cost-efficient than creating replicas on every node of the path and yield higher utilized replicas than client side caching. Traffic hubs are determined as the nodes where multiple data access paths for a file coincides. The algorithm is decentralized and self-adaptive in the sense that each node can decide

Table 2.3 : Summary of the literature on decentralized replica placement [2].

	Decentralized	Partial Information	Dynamic	Environment	Topology	Objectives
[90]	✓			Cloud-CDN	Unrestricted	Proximity
[91]	✓		✓	Cloud	Complete	Prox., Cost, Bandwidth, Avail.
[92]	✓		✓	Data Grid	Unrestricted	Prox., Cost, Bandwidth, Avail.
[93]	✓		✓	P2P	Unrestricted	Cost, Bandwidth, Availability
[94]	✓		✓	Web	Unrestricted	Proximity, Bandwidth
[95]	✓	✓		N/A	Multi-Tier	Proximity, Bandwidth
[88]	✓	✓	✓	Web	Unrestricted	Prox., Cost, Load B., Bandwidth
[89]	✓	✓	✓	P2P	Unrestricted	Proximity, Cost
[87]	✓	✓	✓	Web	Unrestricted	Proximity, Cost
[96]	✓	✓	✓	Web	Unrestricted	Proximity, Cost
[2]	✓	✓	✓	Cloud	Unrestricted	Proximity, Cost, Bandwidth

whether to store replicas. They do so by analyzing the traffic running through them and determining the most popular files. Although this is possible in the P2P file sharing scenario, it would cause a privacy issue in the cloud environment.

Authors of article [95] suggest a distributed replica placement algorithm for systems that consist of predefined replication groups (e.g. departments in a university). When a data is requested, it can be fetched from other servers in the group that would result in shorter access time than fetching from the origin server. However, the algorithm is not applicable to Cloud or Edge Computing models where inherent replication groups do not exist and the topology is immense as well as highly dynamic.

Data grid topology is partitioned into network regions in [92] and replication and placement decisions are optimized at each region with a focus on network congestion avoidance. Similarly, authors of the paper [90] suggest partitioning the cloud topology graph and greedily deciding the number and location of replicas at each cluster. The objective is to minimize the number hops between the content provider and its users. Although the replica placement is conducted in a distributed manner in these two approaches, complete topology information is still required for the centralized partitioning phase.

One of the most similar data replication approaches to the one in this thesis is [91]. Authors propose an algorithm based on a game-theoretical model which is executed at each node autonomously. The algorithm may replicate or migrate the data at the node or remove it depending on the availability of the file, as well as communication and monetary costs. The method mainly differ in the sense that each node is aware of locations of other replicas as well as the rent price, demand, and bandwidth of all other nodes. Moreover, their knowledge must be periodically updated which would cause a performance bottleneck in the Edge computing scenario with hundreds, if not thousands of nodes.

Primal-Dual based distributed approximation algorithms for the FLP are present in the graph theory literature [97, 98] and their variations are recently applied to the problem of Internet service placement [87, 96]. In the former, a small scale FLP is solved iteratively for each group of nodes that are in close proximity (called an r-ball where r is the maximum number of hops). The latter introduces a conditional betweenness centrality metric and removes the restriction that the facility location is within the r-ball. To that end, the most central nodes in the topology are calculated and the FLP is solved on this selected subset of nodes at each iteration.

Contrary to these two approaches, the aim in this thesis is to eliminate any message passing or broadcasting between non-adjacent nodes. Reporting the demand estimates, centrality values or facility location decision over multiple hops is acceptable for a single instance FLP case such as replicating an Internet Service. However, the number of data objects and thus the number of FLP instances can easily reach thousands or even millions which would cause an infeasible network overhead. Hence, any replica placement algorithm which requires frequent communication between the nodes is impractical for granular data objects.

3. SYSTEM MODEL

Although real-world test-beds are usually preferable over simulations, evaluation via simulation is more appropriate for the specific case of distributed cloud due to the following reasons.

- Since the standardization of distributed clouds is not finalized yet, adhering to a certain standard in terms of inter-cloud communications could turn out to be a misleading direction as the technology matures. This would put the generality of the results at risk.
- A geographically distributed and federated cloud environment that is open to scientific community is not available currently. Hence it would not be possible to observe the widely distributed characteristics of a federated cloud on a real-world infrastructure.

Therefore, a framework for modeling, simulating and evaluating resource mapping in the distributed cloud environment is developed as a part of this thesis. RalloCloud [1] is developed on top of the popular cloud simulation toolkit, CloudSim [99]. Experimental evalutation of the algorithms proposed in Chapters 4 and 5 are conducted on the RalloCloud framework. The rest of this chapter explains how cloud infrastructure, VM clusters, data objects and resource mapping problems are modeled as well as the performance criteria to evaluate mapping algorithms. CloudSim does not natively support the introduced features.

3.1 Problem Modeling

3.1.1 Entities

Components of a real-world cloud service and cloud infrastructure need to be modelled in the simulation framework so that the resource mapping can be simulated realistically. Typically, a cloud service consists of several VMs, a topology and a database, while the infrastructure consists of data centers and network connections. The relations between the entities in RalloCloud which are visualized with a entity-relationship model in Figure 3.1, are also taken from the real-world cloud brokerage scenario.

A **cloud data center** (cloud for short) is defined with its resource capacities (CPU, memory, storage) and its geographical location. **Physical topology** is the network connections between clouds which include bandwidth capacity and latency. Each cloud is attached to a **cloud broker** that is responsible for; (1) receiving user requests in the form of VM clusters and allocating resources for them, and (2) receiving data requests of those VMs and replicating data objects accordingly. It is possible for a broker to dispatch VMs and replicas to other brokers. This federated environment allows to model realistic cloud brokerage scenarios.

The **user**, requests a VM Cluster and a database for the cloud-based service. A **VM cluster**, on the other hand, is composed of **VMs**, a **virtual topology** and the geographical location of requesting user. Each VM is defined with its size and resource requirements (CPU, memory, storage) while the virtual topology is the collection of required bandwidths between VMs. A VM cluster starts executing when each VM in that cluster is deployed to a cloud and it is terminated when each VM in that cluster completes its execution.

Both the VM cluster and the cloud federation are represented with weighted undirected graphs; $G_C = (V_C, E_C, A_C^V, A_C^E)$ and $G_F = (V_F, E_F, A_F^V, A_F^E)$, respectively. V is the set of vertices and E is set of edges while A^V is the attributes of the vertices and A^E is the attributes of the edges. Subscripts C and F as used to distinguish cluster and federation. A^V and A^E consist of the attributes that are as explained above, for instance, A_F^E defines the bandwidth capacity and latency of each network connection in the federation.

Finally, the **database** of the cloud service is composed of **data objects** with varying sizes. The database is stored in a certain cloud which provides storage as a service. **Replicas** of the data objects can be created and stored in other clouds. VMs may require multiple data objects during their execution depending on the behaviour of their user base. Such requirements need to be answered from either the original data object or one of its replicas.

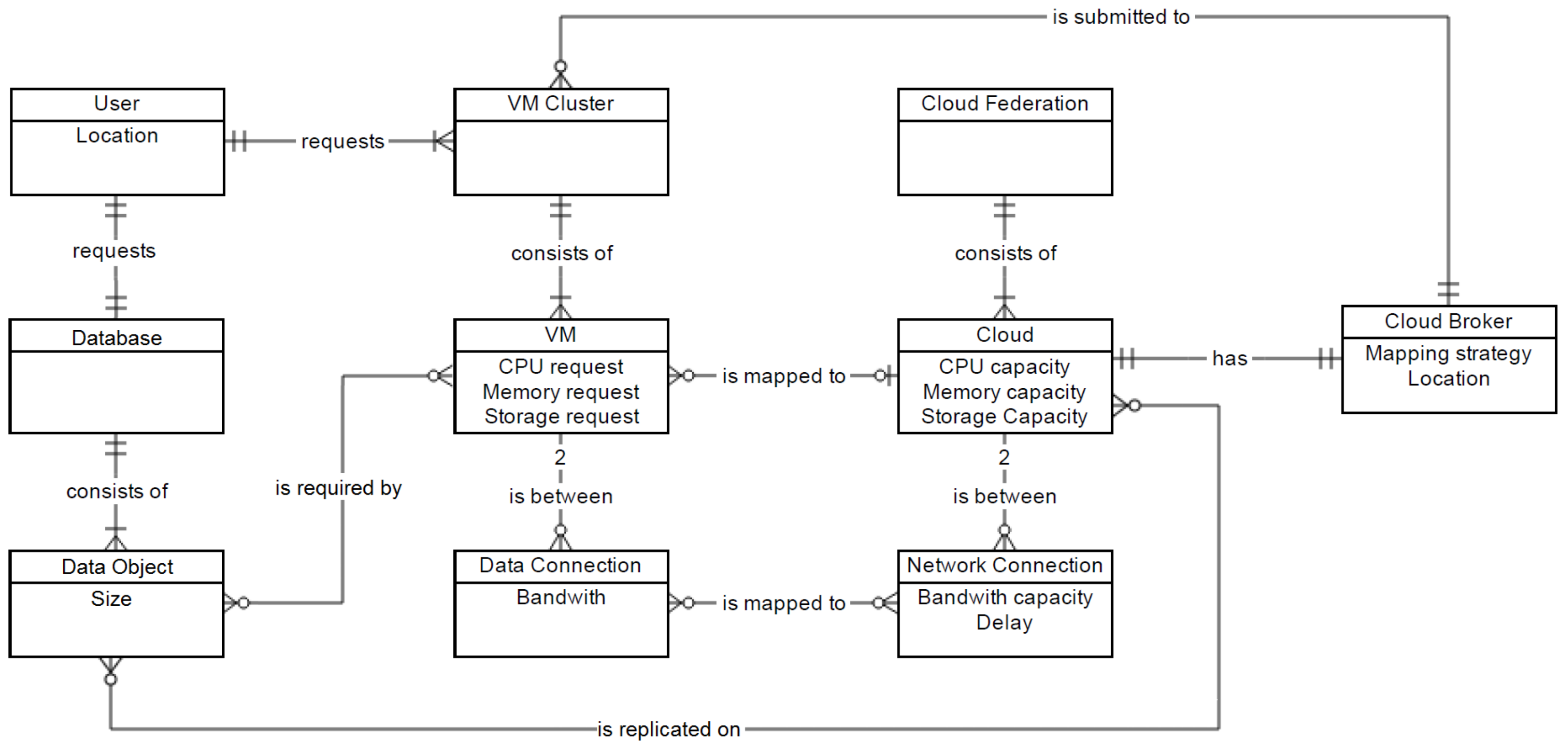


Figure 3.1 : Partial entity-relationship model of ReIloCloud.

3.1.2 Topology mapping problem

One of the two problems considered in this thesis is to efficiently and optimally map user VMs to distinct cloud providers. Embedding a user's request of a VM cluster onto the cloud federation involves mapping each vertex $v \in V_C$ to exactly one vertex $v' \in V_F$ and mapping each edge $e \in E_C$ to exactly one set of edges $E' \subseteq E_F$. Three conditions must hold for the two mappings to be considered a valid embedding:

1. Remaining resource capacities of all v' and $e' \in E'$ are greater than or equal to the sum of required resources by all v and e which are mapped to v' and E'.
2. For each e, mapped E' forms a path between the two distinct vertices v'_1 and v'_2 to which endpoint vertices of e (v_1, v_2) are mapped.
3. If multiple vertices $v_1, \cdots, v_n \in V_C$ are mapped to the same v', then any e which has both endpoint vertices in $v_1, \cdots, v_n$ is mapped to $E' = \varnothing$.

In the model, it is possible to map multiple VMs to a single cloud as long as the resource capacity is enough. Similarly, a network connection can be utilized by multiple data connections and multiple VM clusters can be embedded onto the cloud federation.

3.1.3 Replica management problem

The first aspect of replica management is to find the optimum number and location of replicas so that the latency and cost are minimized. FLP is adapted in order to obtain a general objective function for these two conflicting goals. In replica placement, the optimum number of replicas cannot be known a priori. Thus, the cost function in equation 2.4 better fits the case. In addition, only the uncapacitated FLP is considered since the allocation of cloud resources is already handled in the topology mapping.

FLP models with multiple products (i.e. multi-commodity FLP) aim to determine facility locations which optimize distance to demands for all products. This is critical for industrial cases where building a facility has a fixed cost and it is infeasible to build a different facility for each product [14]. However, in the IaaS scenario, upfront costs and commitments are eliminated and the customer only pays for the storage used. Thus, it is not necessary to cluster multiple replicas in a provider and the location of each replica can be optimized individually. Hence, a single-product FLP instance is solved for each replica.

Facility opening cost (f_j) in equation 2.4 translates to storage cost that would be charged by the IaaS provider for the replica until the next iteration of the algorithm (during the next epoch).

$$f_j = unit\_price_j \cdot replica\_size \cdot epoch \tag{3.1}$$

Customer demand (h_i) for a replica is proportional to the number of requests received for that data during the previous epoch.

$$h_i = num\_requests_i \cdot replica\_size \tag{3.2}$$

Finally, distance metric d_{ij} is chosen as the latency between a virtual machine and the replica location. Since f_j is a monetary unit, d_{ij} should also be converted to the same unit so that two sides of the addition are commensurable. Thus, a unit conversion factor, λ, is suggested to represent the expendable unit cost in exchange for a unit decrease in latency per unit demand. Value of λ also determines the level of replica expansion.

$$d_{ij} = latency_{ij} \cdot \lambda \tag{3.3}$$

Final definition of the cost function for the adapted FLP is presented in equation 3.4. Note that, *replica_size* is removed from the both sides of the addition since it is fixed for each problem instance and has no effect on the minimization objective. Here Y_j indicates that a replica is stored at node j while X_{ij} indicates that the replica at node j has the least latency to the VM.

$$Total\ Cost \ = \sum_j unit\_price_j \cdot epoch \cdot Y_j \ + \sum_i \sum_j num\_requests_i \cdot latency_{ij} \cdot \lambda \cdot X_{ij} \tag{3.4}$$

It is assumed that the precise knowledge of current demand for each data object and storage prices of IaaS providers will be available. There also exists strategic FLP variations which model the uncertainty in future demands and costs. They place facilities as long-term investments which will be more vulnerable to such changes because building a facility is a cost- and time-critical decision. Again, in the case of cloud computing such constraints are no longer valid. By favour of pay-as-you-go storage, replicas can be relocated momentarily for little or no cost. Service interruption can be avoided by keeping the replica at the source node until it is migrated to the destination.

The second aspect of replica management considered in this thesis is replica discovery. As explained in detail in Chapter 5, it deals with the awareness of closest replicas by cloud data centers. Without an effective replica discovery, request would be sent to central storage even when nearby replicas are available.

3.1.4 Network modeling

Network connections in the cloud infrastructure are represented with the bandwidth capacity and latency. Bandwidth is modeled a constraint that limits the amount of data connection that can utilize a network connection. If two connected but not adjacent (i.e. $|E'| > 1$) entities communicate, then the bandwidth of all network connections of the communication path are utilized the same amount which is equal to the size of requested data connection. Latency, on the other hand, is not modeled as a constraint but rather a factor that affects the performance of the cloud services and a performance criteria. Three types of latency are considered in RalloCloud regarding the transfer of VMs and data.

Deployment Latency determines the time it takes to deploy a VM after it is mapped to a certain cloud and it is effective only in the initialization phase of the VM. It is correlated to VM size (S) and the sum of the latencies (L) on the shortest path (P) between a VM and the user it is submitted by. It is also inversely correlated to the allocated bandwidth (B) on the path. Even if the selected cloud is in the same location as the user, there is a minimum deployment latency (M) experienced due to VM creation.

$$Deployment\ Latency = M + \sum_{i \in P} L_i + \frac{S}{B} \tag{3.5}$$

Communication Latency is between the communicating VMs that form a cluster and, it is in effect after the deployment of VMs, contrary to deployment latency. It has an impact on the execution time of the VM that is correlated to observed latency and the size of transferred data (D) and inversely correlated to the allocated bandwidth (B). Observed latency between two VMs is the sum of latencies (L) on the path (P) to which their data connection is mapped. If two VMs are deployed to the same cloud or there is no data connection, then the communication latency between them is negligible.

$$Communication\,Latency = \sum_{i \in P} L_i + \frac{D}{B} \tag{3.6}$$

Data Access Latency is a specific form of communication latency. It is always between a VM and a copy of a data object. When the a user action or programmatic decision necessitates a data object to be fetched to the VM, the object is requested and a certain communication latency is experienced unless it is available locally. Data access latency is calculated the same way as the communication latency (Equation 3.6).

3.1.5 Cost modeling

RalloCloud supports a dynamic pricing policy called Trough Filling [100] for IaaS providers in addition to the fixed pricing. The policy is based on yield management strategy in economics, and aims to maximize revenue from a limited and perishable resource. In this case, the price of a certain cloud resource (CPU, memory, storage or bandwidth) varies directly with its utilization, that is, the less percentage of the resource is utilized, the lower is the unit price of that resource. It should be noted that utilization of a resource in a cloud affects its price only in that cloud, not in the whole federation.

Unit cost of a VM is the sum of its reserved resources multiplied by their unit prices at the time and location of its deployment. Similarly, unit cost of a network connection is the reserved bandwidth multiplied by the unit bandwidth price and the unit cost of replica storage is the replica size multiplied by the unit storage price. Then, total cost of a cloud service is the sum of unit costs of its VMs, network connections and replicas, multiplied by their utilization duration as shown in Equation 3.7.

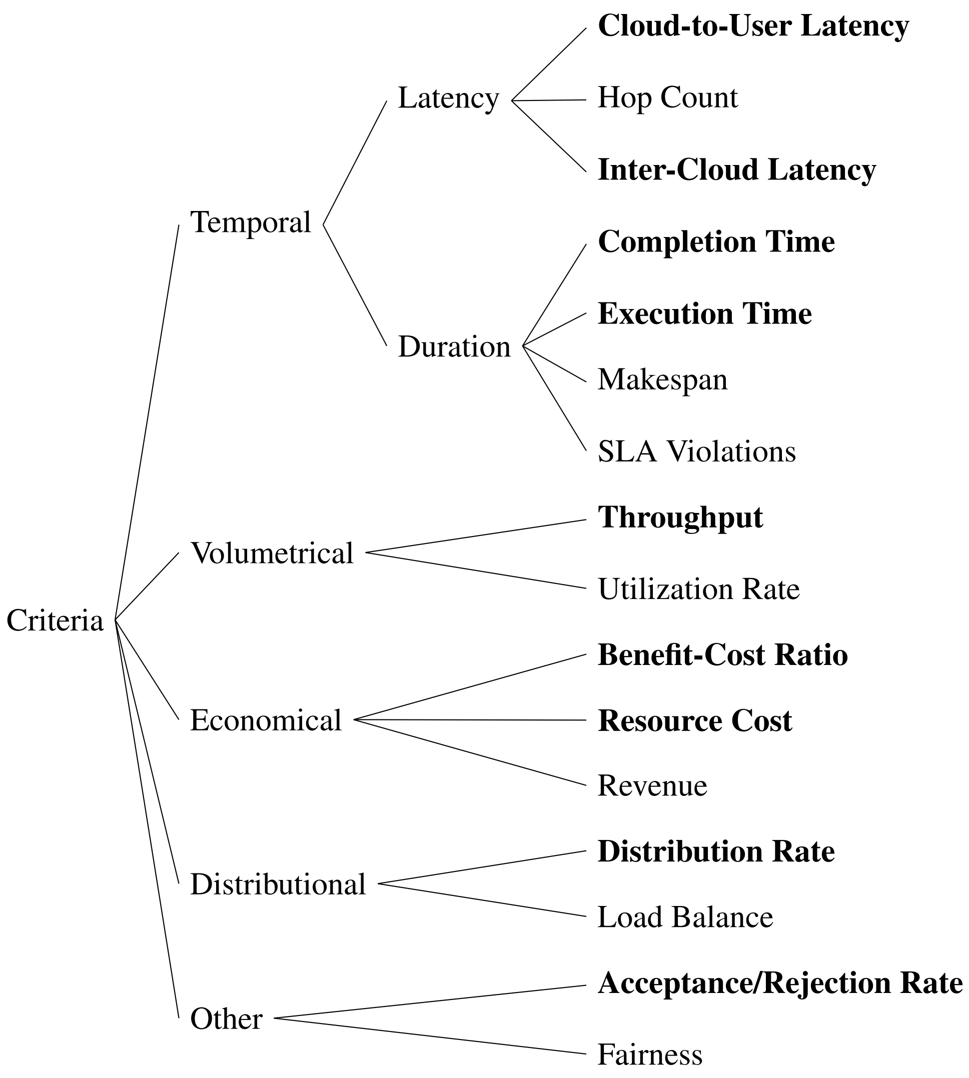


Figure 3.2 : Hierarchical categorization of the performance criteria.

$$\begin{aligned} Total\ Service\ Cost = & \sum_{i \in V_C} \sum_{j \in A_C^V} resource\_size_{ij} \cdot unit\_price_{ij} \cdot duration_{ij} \\ & + \sum_{i \in E_C} resource\_size_i \cdot unit\_price_i \cdot duration_i \\ & + \sum_i replica\_size_i \cdot unit\_price_i \cdot duration_i \end{aligned} \tag{3.7}$$

3.2 Performance Criteria

RalloCloud framework contains a module for measuring performance criteria to evaluate the quality of mappings and compare algorithms. In order to idenfy criteria to include related literature is reviewed, several criteria are identified, some new ones are suggested and all of them are categorized as given in Figure 3.2. To keep the presentation concise and due to the fact that evaluation results of some criteria came out similar, 9 criteria are chosen (marked in bold). The selection carefully includes at least one criterion from each category. Definition of the selected criteria are provided below.

Cloud-to-User Latency is measured between a VM cluster the user who initiates it. It affects the deployment latency of the VM. Cloud-to-User Latency can be calculated as is Equation 3.8 where p_1 is the shortest path between the user and the cloud.

$$Cloud-to-User\ Latency() = \sum_{i \in p_1} Latency(i) \tag{3.8}$$

Inter-Cloud Latency is measured either among the VMs that constitute a VM cluster or between a VM and a replica location. It affects the communication and data access latency of the VM. Inter-Cloud Latency can be calculated as is Equation 3.9 where p_2 is the shortest path between the two nodes that contain the components.

$$Inter-Cloud\ Latency = \sum_{i \in p_2} Latency(i) \tag{3.9}$$

Completion Time is the duration between the arrival of a request and its successfully completion. It is normalized by the size of the task executed in million instructions. Completion time is affected by deployment, communication, and data access latency as well as pending time. Pending time is measured when the cluster cannot be deployed immediately because there is no available cloud at that time.

$$Completion\ Time = \frac{T_{complete} - T_{arrive}}{Task\ size} \tag{3.10}$$

Execution Time is the duration between the deployment of a VM cluster and its completion. Also normalized by task size, it is affected by communication and data access latency but not the deployment latency or pending time.

$$Execution\ Time = \frac{T_{complete} - T_{begin}}{Task\ size} \tag{3.11}$$

Throughput is measured as the millions instructions processed in the whole federation per second (MIPS). It can be calculated as is Equation 3.12 where S is the set of all tasks executed.

$$Throughput = \frac{\sum_{i \in S} Size(i)}{max_{i \in S}(T_{complete}(i))} \tag{3.12}$$

Benefit-Cost Ratio is used evaluate the efficiency of algorithms to address the trade-off between data access latency and data storage/transfer cost. It is calculated as latency improvement rate divided by cost increase rate as shown in Equation 3.13.

$$Benefit-Cost\ Ratio = \frac{Rate\ of\ Latency\ Improvement}{Rate\ of\ Cost\ Increase} \tag{3.13}$$

Resource Cost is the total cost of CPU, memory, storage and bandwidth for a cloud service during its execution time. Calculation of the total cost is described is Section 3.1.5.

Distribution Rate is the extent that the VMs of a service are placed to separate clouds. It is measured as the ratio of number of distinct clouds employed for VMs to the number of VMs in the cluster.

$$Distribution\ Rate = \frac{|Clouds|}{|VMs|} \tag{3.14}$$

Rejection Rate is the percentage of VMs that are submitted to a cloud but failed to be deployed due to lack of resources. Such a case may occur if the embedding or matching algorithm is unaware of cloud providers' utilization or if multiple instances of the algorithm submits VMs concurrently to the same cloud. Depending on the algorithm, rejected VMs may be dispatched to other cloud providers or queued.

$$Rejection\ Rate = \frac{|Rejected\ VMs|}{|Accepted\ VMs| + |Rejected\ VMs|} \tag{3.15}$$

4. TOPOLOGY MAPPING

4.1 Problem Definition

Magnitude of the digital data being generated and the speed at which it is aggregating in cloud is enormous. Even the largest IaaS providers may run into a difficulty in scalability in the not so distant future because of this enormous increase in cloud service usage. Moreover, cloud users are geographically distributed and they access the data from all around the world making it increasingly hard to provide a globally consistent QoS. Federated cloud [101,102] is motivated by such dangers and obstacles. It is defined as the mechanisms, policies and technologies to coordinate and unite cloud data centers even if they are managed by different vendors. Cloud providers voluntarily collaborate in the federated cloud scenario as distinct from multi-cloud where multiple independent clouds are utilized by a service uninformedly [103].

Federated clouds allow vendors to easily dispatch load to the other members of the federation, delivering the infinite scalability promise of cloud computing [101]. This improves the QoS by giving cloud vendors the ability to cope with demand peaks as well as to provide complete geographical coverage. Moreover, such an interoperability at the infrastructure level sets cloud users free of vendor lock-in as well as allowing private data center owners to easily hybridize their infrastructure. Finally and more importantly for this thesis, with federated cloud it is possible to scale VM clusters across multiple vendor clouds [102]. It is a common practice to isolate different components of a service (e.g. storage, application logic, user interface) using distinct VMs that communicate among themselves. Here, a VM cluster is a group of collaborating VMs that constitute a 'cloud service'. Ability to deploy cooperating VMs on different clouds provides the following advantages from the point of a cloud based service provider (and an IaaS user).

Availability and Disaster Recovery The effect of a failure or low QoS in a cloud vendor can be easily compensated with minimal damage to the overall service.

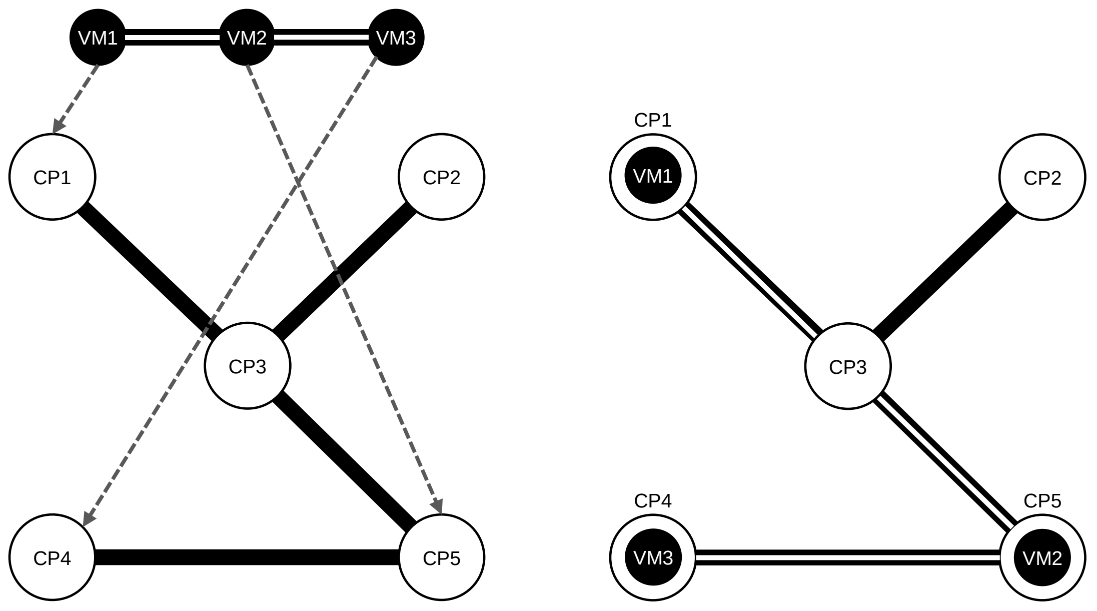
Geographical Coverage Geographically distributed user base of the service can be covered with a high QoS.

Vendor Lock-in Avoidance VMs can be migrated easily and quickly between vendors in case of any dissatisfaction.

Cost Reduction Different pricing policies of the vendors can be exploited to reduce infrastructure cost.

However, distributed placement of VMs onto a federated cloud infrastructure also presents new problems that need to be addressed. One of the most significant of these problems is to develop an efficient mapper between the physical topology and the user requests in the form of virtual topologies [37]. A virtual topology defines the bandwidth requirements for data flows between VMs in the same cluster. A service provider characterizes the amount of data that will be transferred between each VM pair in terms of bandwidth. On the other hand, physical topology defines the available dedicated network connections between cloud providers as well as their bandwidth capacities and latencies. Direct dedicated connections may not exist between all cloud provider pairs in the federation and not all VM pairs in a cluster need to communicate, thus neither of the topologies are complete graphs in general. When adjacent VMs are mapped to nonadjacent clouds, the connection has to be multi-hop, thus latency and bandwidth utilization increases.

Figure 4.1 visualizes the mapping and deployment of a single VM cluster of three VMs onto a federation of 5 cloud providers (CPs). Here, physical topology is represented with white circles (clouds) and thick lines (inter-cloud network connections) while virtual topology is represented with black circles (VMs) and double lines (data flows). According to the requested virtual topology, data transfer will occur between pairs VM1 – VM2 and VM2 – VM3 but not VM1 – VM3. Figure 4.1(a) demonstrates an example mapping between VMs and clouds shown with dashed arrows. Although, different mappings can be generated by optimization algorithms with different objective functions, the mapping relation must satisfy the function property (each VM must be mapped to exactly one cloud). In Figure 4.1(b) VMs are dispatched to clouds according to the mapping in Figure 4.1(a) and deployed there. During the execution, data transfer between VM2 and VM3 will be direct, while it will be through CP3 for



(a) Mapping (b) Deployment

Figure 4.1 : VM cluster embedding [4].

VM1 and VM2. In a real world scenario with non-trivial number of VM clusters, multiple VMs belonging to different clusters would be hosted at each cloud.

VM cluster embedding (VMCE) problem deals with finding a mapping between inter-connected VMs and clouds, as exemplified by Figure 4.1(a). The problem is not trivial due to presence of multiple constraints and objectives [36]. First of all, clouds have limited and heterogeneous capacities in terms of CPU, memory and storage. Similarly, network connections have varied latencies and bandwidth capacities. VMs of different sizes should be placed on clouds respecting such limits and making an efficient use of the resources to increase utilization. A similar problem is referred as Virtual Network Embedding (VNE) in the literature [37,42,104]. Definition of VMCE in this thesis diverges from VNE as it also involves constraints and requirements for nodes (clouds and VMs) in addition to the edges (network).

Network plays a key role in the performance of distributed cloud services. Hence, communicating VMs should be placed on clouds that have low latency inter se so that thedata transfer would be fast and the QoS would be high. Another factor is the latency between the user base and selected clouds. In the case of provisional applications such as scientific calculations or MapReduce [105] jobs, high latency also extends the execution time and accordingly increases resource costs. Better latency optimization is vital for distributed, soft real-time services and applications (e.g. video streaming, online gaming) to be executed on federated cloud. Cloud computing may find a new area of application in real-time software provided that the network related challenges are overcome [106].

A novel VMCE algorithm for federated cloud, Topology Based Mapping (TBM) [4], is proposed in this chapter. TBM employs a graph theoretical approach in combination with greedy heuristics in order to reduce network latency and optimize bandwidth utilization. TBM algorithm mainly focuses on the bandwidth and latency that are (1) within the VM cluster, and (2) between the VM cluster and the intermediate cloud user who submits it (e.g. a cloud-based service provider or a scientist running a high performance job). Moreover, evaluation of the TBM algorithm as well as baseline heuristics in terms of latency, execution time, throughput, cost, rejection rate, etc. is performed and the results are provided.

4.2 Use Case Scenario

Consider a small cloud-based text translation service. The service is deployed on a two-tier architecture where first tier contains the natural language processing (NLP) algorithms and the user interface while the second tier is for persistence of user account information and texts. The service provider wishes to replicate the first tier and host the replica VMs in two different providers in order to achieve the following two benefits: 1) overall latency will be decreased since the users of the service will be served by the closest replica, and 2) the service will continue its execution in case of a failure in one of the replicas and the failed replica can be recovered later (possibly in another cloud) with minimal loss of data.

Replication of the second tier is handled in the Chapter 5 of the thesis, so it would be considered as a single centralized storage for now. However, as the second tier has different resource requirements (see Table 4.1) than the first one, it may be economically beneficial to host it in a separate cloud provider where the storage pricing is more convenient. Moreover, placing the storage to a middle ground between the VMs in terms of latency will provide similar QoS for the users of the both of them. Consequently, a cluster of three VMs need to be deployed in three different clouds for the service. Resource requirements of these VMs are provided in Table 4.1. Here, VM1 and VM3 are for the replicas of the first tier and VM2 is for the second tier. Resource

Table 4.1 : Resource requirements of the VMs for the text translation service.

VM	CPU Cores	Memory	Storage	Bandwidth
VM1	8	30 GB	2 x 80 GB SSD	1000 Mbps out
				500 Mbps to VM2 (Dedicated)
VM2	4	30.5 GB	3 x 2 TB HDD	500 Mbps to VM1 (Dedicated)
				500 Mbps to VM3 (Dedicated)
VM3	8	30 GB	2 x 80 GB SSD	1000 Mbps out
				500 Mbps to VM2 (Dedicated)

capacities of VMs in this scenario are taken from Amazon Web Services [107] EC2 dedicated instances m3.2xlarge and d2.xlarge.

The service needs 500 Mbps dedicated bandwidth between VM1 and VM2 as well as between VM2 and VM3 as shown in the bandwidth column of Table 4.1. There is no requirement for bandwidth between VM1 and VM3. Thus, the virtual topology of the VM cluster is as given in Figure 4.1(a). In addition, being the interface of the service, VM1 and VM3 requires 1000 Mbps bandwidth-out each for the user interaction. For simplicity, let us also assume that all five cloud providers in Figure 4.1(a) have enough available resources to accept any of these VMs. The assumption is for this example only and not valid in the actual algorithm.

An arbitrary mapping between VMs and clouds would result in high latency data connections between VM1 and VM2 or between VM2 and VM3. For instance, consider the mapping in Figure 4.1(b), there is no direct and dedicated network connection between CP1 and CP5 so the latency between VM1 and VM2 is high. This would decrease the QoS for the users who are served by VM1 because of the delay perceived when accessing the storage. Moreover, it will increase the execution time of the NLP algorithms and thus the cost of infrastructure. From the cloud providers (CP3) point of view, such a mapping is a waste of bandwidth capacity which could have been leased to another customer.

The TMB algorithm, on the other hand, would decrease the latency between VM1 and VM2 by mapping VM1 to CP3 instead of CP1. Because the subgraph consisting of CP3, CP4, CP5 and their inter-connections is isomorphic to the topology graph of the requested VM cluster as denoted by the bijective function f in Equation 4.1.

$$f = (\text{VM1} \mapsto \text{CP3}, \text{VM2} \mapsto \text{CP5}, \text{VM3} \mapsto \text{CP4}) \tag{4.1}$$

Naturally, the algorithm considers several other factors and conditions than this trivial example in mapping three VMs to clouds. These are elaborated in the following sections. Motivations of the TBM algorithm, some of which are demonstrated in this use case, are as follows:

- Reducing the average inter-cloud latency by placing communicating VMs to locations that are connected with dedicated network connections.
- Reducing the average cloud-to-user latency by prioritising subgraphs with low latency to user.
- Improving the QoS and decreasing the execution time as a result of low latency communication.
- Decreasing the resource cost for the service provider and increasing the throughput (and the profit) for the cloud provider.
- Finding effective, “good enough" mappings via heuristics when the optimal solution (isomorphic subgraph) is not available.

4.3 Proposed Solution

Efficient utilization of the network infrastructure is crucial for the VMCE problem due to the following reasons: 1) network latency affects execution time and cost, and 2) availability of bandwidth affects the acceptance rate of new requests. Moreover, both these factors also contribute to overall utilization, throughput and revenue of the federation. Hence, the main factors that TBM algorithm is built to optimize are latency and bandwidth. It is observed that decreasing latency and efficient use of bandwidth are neither conflicting, nor completely parallel objectives.

The main idea behind the TBM is to map a VM cluster request to a subset of clouds in the federation that has the same (or at least similar) network topology as the request. To achieve this, TBM algorithm searches for subgraphs of G_F that are isomorphic to G_C. If such a subgraph does not exist or if the mapping is not valid, it deducts to a heuristic approach to find a homeomorphic subgraph instead. Each cloud broker in the federation runs the algorithm locally for each incoming VM cluster request and dispatches VMs to the other clouds according to the matching. In order to gain the

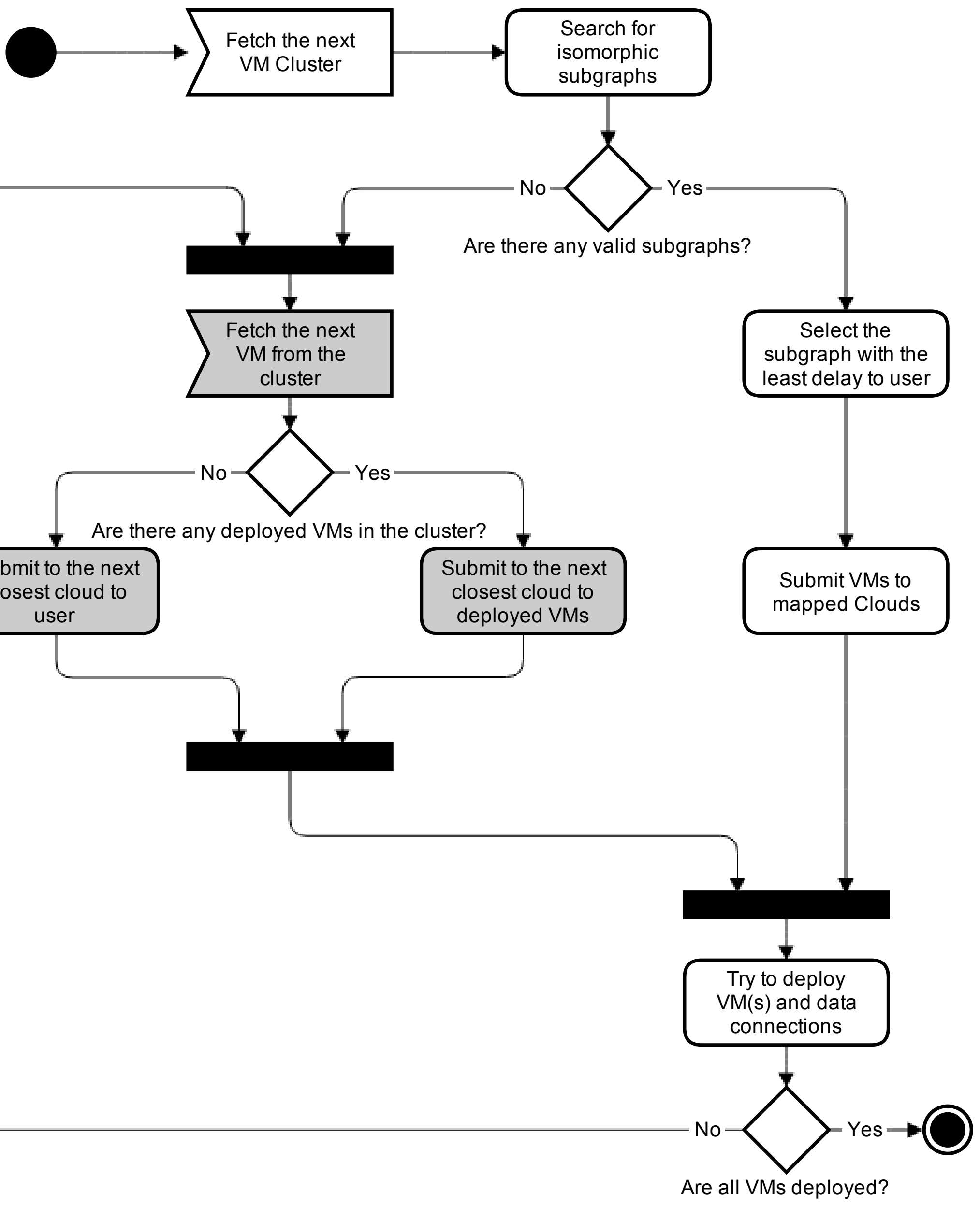

Fetch the next VM Cluster
Search for isomorphic subgraphs
No
Yes
Are there any valid subgraphs?
Fetch the next VM from the cluster
Select the subgraph with the least delay to user
No
Yes
Are there any deployed VMs in the cluster?
Submit to the next closest cloud to user
Submit to the next closest cloud to deployed VMs
Submit VMs to mapped Clouds
Try to deploy VM(s) and data connections
No
Yes
Are all VMs deployed?

Figure 4.2 : UML activity diagram of the TBM algorithm [4].

benefits of the distributed VM placement that are explained in Section 4.1, TBM tries to map each VM to a different cloud.

Figure 4.2 demonstrates the flow of the algorithm where the main scenario is represented with hollow rectangles and alternative heuristic part with the shaded ones. In addition, a high level pseudo code is given in Algorithm 2. Here, lines 1 to 6 are the main scenario and 7 to 14 are the heuristic part.

Inputs to the TBM algorithm are listed below.

1. Overall utilization and capacity of each resource in each cloud data center.

Algorithm 2: Pseudo code of the TBM algorithm.

```
Input  : List $Q$ of queued clusters requests $G_C = (V_C, E_C, A_C^V, A_C^E)$
         Topology $G_F = (V_F, E_F, A_F^V, A_F^E)$ of the federation
         Location $l$ of the user
Output: Mapping $M$ between VMs and clouds
1  foreach cluster request $G_C \in Q$ do
2      subgraphs[ ] ← SearchIsomorphicSubgraph($G_F$, $G_C$)
3      if size(subgraphs[ ])>0 then
4          chosenSubgraph ← argmin$_x$(AvgLatency(subgraph[x], l))
5          foreach virtual machine $VM \in V_C$ do
6              $M[VM]$ ← corresponding node in chosenSubgraph
7      else
8          foreach virtual machine $VM \in V_C$ do
9              deployedVMs[ ] ← deployed($G_C$)
10             if size(deployedVMs[ ])>0 then
11                 chosenNode ← argmin$_x$(AvgLatency($V_F$[x], deployedVMs[ ]))
12             else
13                 chosenNode ← argmin$_x$(AvgLatency($V_F$[x], l))
14             $M[VM]$ ← chosenNode
```

2. Bandwidth utilization and capacity as well as average latency of each network connection between cloud data centers.

3. Resource requirements of each VM in the requested clusters.

4. Bandwidth requirement between each VM pair.

5. Location of the cloud user

4.3.1 Subgraph isomorphism based mapping

The first step of the algorithm is to fetch the VM cluster request and start an isomorphic subgraph search on the federation topology. If the search ends up with exactly one valid mapping, VMs are submitted to their matched clouds. If there are multiple candidates, however, alternatives are sorted by average cloud-to-user latency and the VMs are submitted to the subgraph with the least average latency. Algorithm is completed if all VMs are successfully deployed to the mapped clouds.

Local All Different (LAD) filtering procedure [13] (explained in Subsection 2.1.1) is implemented for the efficient discovery of isomorphic subgraphs. It helps to reduce the search space for subgraph matching by detecting and pruning branches that do not contain solutions. Given a pair of nodes (n_p, n_t) from the pattern and target graphs, LAD filtering builds a bipartite graph where the two sets are the neighbours of n_p and n_t. Two nodes from each set are adjacent if the matching of these two nodes are not previously pruned. On that graph, it checks whether a bipartite matching exists such that all neighbours of n_p are covered. If there does not exist such a matching, then n_p and n_t are incompatible and all the branches that match them are pruned from the search tree.

The dominant part of the TBM algorithm in terms of time complexity is the isomorphic subgraph search. Thus, the complexity of TBM is equal to the complexity of the method used for this search. In the case of LAD filtering, time complexity is $\mathcal{O}(|N_p| \cdot |N_t| \cdot d^4)$. Here $|N_p|$ and $|N_t|$ are the number of nodes in the pattern and target graphs, respectively while d is the greater of the maximal degrees of these two graphs [13]. Although, the complexity can cause a performance bottleneck for large number of nodes, the number of clouds is not expected to exceed few hundreds and each cloud application is expected to be contain at most dozens of VMs in this thesis's usage scenario. Our experimental result in subsection 4.4.3.5 demonstrate that the algorithm finishes within seconds for realistic configurations.

4.3.2 Heuristic mapping

TBM fails to deploy VM cluster to an isomorphic subgraph in two exceptional cases in the main mapping scenario:

No mapping: There does not exist any isomorphic subgraphs of clouds that are valid. In a valid mapping all subgraph nodes (clouds) must hold enough available resources for the VM assigned to them.

Partial deployment: There exists at least one valid subgraph but some of the VMs are rejected by the clouds they are mapped to. Since the algorithm has multiple instances running in each broker, two or more brokers may concurrently decide to submit VMs to the same cloud provider. If the cloud does not hold enough available resources to deploy all these VMs, it rejects the VMs arriving after its full utilization. Consequently, not all VMs of the cluster can be deployed.

TBM switches to the heuristic mode in both cases and tries to find a homeomorphic graph with low latency. In this mode, non-deployed VMs of the cluster are handled separately. Heuristic fetches an arbitrary VM from the cluster and checks whether there exists any other VM in the same cluster that is successfully deployed to a cloud. If that is the case, the fetched VM is submitted to an available cloud that would result in least average inter-cloud latency. However, if all VMs of the cluster are yet to be deployed, then the VM is submitted to the cloud with the least cloud-to-user latency.

To illustrate the heuristic mapping part of the TBM, let us consider that a cluster with 3 VMs. If no mappings can be found for the requested topology, these 3 VMs will be matched to the clouds separately. The VM that is considered in the first place will be submitted to an available cloud with the least latency to the user. The next one will be submitted to an available cloud with the least latency to the first VM and finally the third VM will be submitted to an available cloud with the least average latency to the other two VMs. Since available clouds may be multiple hops away, the subgraph of the federation topology to which the VMs are mapped would not necessarily be isomorphic to the requested VM topology but they would be homeomorphic.

4.3.3 Within cloud mapping

Fundamentally, the problem of optimally mapping VMs to physical machines (PMs) in single cloud data center can be formulated as an NP-hard multidimensional bin packing problem and solved via optimization techniques such as linear programming [9]. However, it would be computationally expensive to run the optimization algorithm at each VM request especially for more than a few PMs. Moreover, optimization would incur large number of VM migrations. Suggested solution is to distribute VMs to PMs using an intelligent heuristic until the point that a migration is inevitable (i.e. when no PM has enough capacity to accept the incoming VM). After this point, an optimization technique can be employed to rearrange the VMs.

A high level pseudo code of the two-part solution is given in Algorithm 3. In part I (lines 4 to 12), migrations are not allowed, and each arriving VM is assigned to the PM determined by an heuristic of evenness. Goal of this part is to keep the utilization of four resources in a VM approximately even. Overall utilization of the resources are maximized by increasing the resource evenness of all VMs [56].

Algorithm 3: Pseudo code for the VM placement strategy

```
Input  : Set $V$ VMs to be assigned
         Sets $P$ of available PMs
Output: Mapping $M[]$ of VMs to PMs
1  rejected ← false
2  while rejected = false do
3      receive next VM $v \in V$
4      assignable ← false
5      foreach PM $p \in P$ do
6          if p has enough capacity for v then
7              assignable ← true
8              $M[v] \leftarrow p$
9              $U[p] \leftarrow$ evennessHeuristic($p, U$)
10             $M[v] \leftarrow \emptyset$
11     if assignable = true then
12         $M[v] \leftarrow \text{argmax}_x(U[x])$
13     else
14         Optimize MIP formulation
15         if optimization succeeds then
16             $M[v] \leftarrow$ MIP output
17         else rejected ← true
```

As an example, if all CPU intensive VMs are assigned to the same PM, its CPU capacity will be fully utilized while other resources are still under-utilized. Due to the lack of idle CPU capacity, no new VM can be assigned to the PM, resulting in waste of idle memory, bandwidth and storage capacities as well as low overall utilization. If different types of VMs are combined effectively on a PM instead, utilization of all four resources will increase at roughly equal rate and it will be able to host more VMs.

To decide which PM is the best for a given VM, it is transiently considered to be assigned to the PMs that has enough remaining capacity one by one and evenness is calculated. Then, the VM is actually assigned to the PM with the best evenness value (line 12). At one point of the part I, incoming VM's resource request will not fit into the remaining capacity of any PM (the VM is not directly assignable and condition in line 11 is not satisfied). This saturation point is not necessarily the optimal VM placement since the heuristic is not aware of the future demands and only approximates to an optimal solution. That means, however, rejected VM can still be assigned to a PM if some other VMs are relocated. Hence the part II of the algorithm which is the optimization, starts at this point.

Part II (lines 13 to 16) runs a MIP formulation solver to optimize placement. It migrates some of the VMs and tries to make room for the new ones. If part II succeeds to assign the VM (line 16), part I of the algorithm continues to receive new VMs; otherwise, it is certain that there is no placement that can assign received VMs to the PMs and the algorithm terminates (line 17).

Part II of the algorithm deals with a NP-hard problem, so for the time complexity analysis only part I is considered. Time complexity of part I is $\mathscr{O}(n) \times \mathscr{T}(m)$ where n is the number of PMs, m is the number of resources and $\mathscr{T}(m)$ is the time complexity of unevenness calculation. $\mathscr{T}(m)$ can be $\mathscr{O}(m^2)$ for some heuristics, however the number of resources is usually a small constant number, thus unevenness is calculated in constant time. Consequently, part I of the algorithm takes linear time.

4.3.3.1 MIP formulation

To determine an optimal VM placement scheme, a Mixed Integer Programming (MIP) formulation is optimized. The number of migrations is minimized when determining the place (PM) of each VM from the pool of PMs. In achieving this, for each solution the place of each VM is compared to its former place and the number of VMs that don't change their place is maximized.

When using MIP, it is required to build formulae that represents the constraints of the system and an objective function to be optimized during the process. In constructing such formulae variables that represent the overall properties of the system shall be used. Below, the variables that are used in the model and their meanings are explained.

- #PMs: Number of PMs present in the cloud environment.
- #VMs: Number of VMs to be placed upon the present PMs.
- oldAsgn: A boolean matrix that holds the present assignment of each VM on a PM.
- newAsgn: A boolean matrix that holds the resulting assignment by MIP.
- resNeed: An integer matrix that holds the amount of resource needed by each VM for the four different type of resources mentioned before.
- resAv: An integer matrix that holds available resources for each PM and each resource.

The objective function given in Equation 4.2 is maximized. In the equation *i* index is used to select among PMs and *j* index is used to select among VMs. Since a VM is either assigned or not to a PM, boolean variables are used. The sum of old and new VM–PM assignment products which produce a value of 1 if the assignment didn't change and 0 if a migration is present are maximized.

$$\sum_{i=1}^{\#PMs} \sum_{j=1}^{\#VMs} (\mathrm{newAsgn}[i][j] \times \mathrm{oldAsgn}[i][j]) \tag{4.2}$$

A number of constraints are also applied in order to drive the MIP to produce valid results. In Equation 4.3 it is guaranteed that each VM is assigned to exactly one PM. Additionally in Equation 4.4 resource needs of each VM that is assigned to a specific PM, are summed up to be less than the available resource assigned to the PM so that no PM is overloaded.

$$\bigwedge_{i=1}^{\#PMs} \left(\sum_{j=1}^{\#VMs} \mathrm{newAsgn}[i][j] = 1 \right) \tag{4.3}$$

$$\bigwedge_{i=1}^{\#PMs} \bigwedge_{j=1}^{\#Res} \left(\left(\sum_{k=1}^{\#VMs} \mathrm{newAsgn}[i][k] \times \mathrm{resNeed}[j][k] \right) \leq \mathrm{resAv}[i][j] \right) \tag{4.4}$$

4.3.3.2 Evenness heuristic

Standard deviation (SD) is a natural choice to find the unevenness of data points since it shows the amount of dispersion from the average. *SD* of the resource utilization on a VM *v* can be calculated as follows.

$$SD(p) = \sqrt{\sum_{i=1}^{m} (r_i - \bar{r})^2} \tag{4.5}$$

Here, r_i denotes the utilization of the *i*th resource, while $\bar{r}$ is the average utilization of all resources of *p*. *m* is the number of resources. For the trivial case of only four instances, a simpler heuristic may be used. Since the focus is more on outliers than inliers, difference between the maximum and minimum utilization rates (span or SP) can be a good candidate for an evenness heuristic.

$$SP(p) = \max_{i \in [1,m]} (r_i) - \min_{i \in [1,m]} (r_i) \tag{4.6}$$

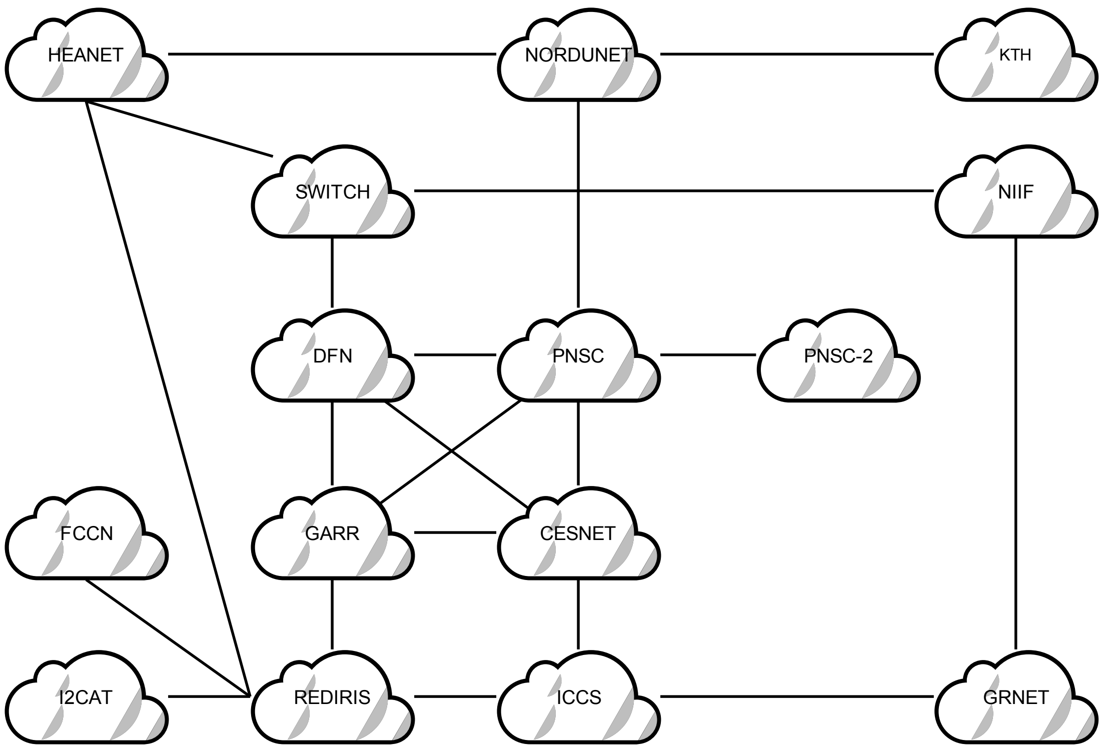


Figure 4.3 : Physical infrastructure topology [5].

4.4 Evaluation

4.4.1 Experimental setup

The simulations are carried out on the ealloCloud framework explained in Chapter 3. VM cluster topologies and sizes may vary greatly in cloud systems [103]. In complicated cases, number of VMs can easily reach to factors of ten. On the other hand, if a medium size web application is considered, it may consist of a few VMs for persistence layer, user interface, etc. Even smaller applications may need only one VM. Without loss of generality, VM clusters are generated with VM count based on a Poisson distribution with a mean of three. Data connection topology of the generated clusters are chosen uniformly at random among; complete, linear, circular and star.

Federation topology, on the other hand, is taken from a real-world example: an experimental networked cloud infrastructure proposed in [5]. Implemented version of the topology (Figure 4.3) includes 14 point of presences across Europe and up to 4 virtualization servers (physical machines) at each location. There is also another node (NORDUNET) which is included to the topology but does not contain a data center.

The architecture follows the IaaS paradigm and the cloud capacities are heterogeneous in proportion to the number of virtualization servers at each point of presence.

Four simulation variables and their effect on the embedding quality are evaluated. Although ValloCloud and TBM support CPU, memory and storage resources, **VM size** is represented with only memory requirement of VMs for simplicity. It determines the number of VMs that can be deployed to a single cloud. Within cloud resource allocation and the minimum span heuristic is evaluated separately. In the simulation, $64x$ of memory is assigned to each virtualization server and $1x$ to $8x$ of memory to each VM. Then, **bandwidth size** of data connections between VMs is up to $8y$ and each network connection has a bandwidth capacity of $80y$. Another simulation variable, **I/O data length** is the amount of data that is transferred between communicating VMs in a cluster. It is between $1z$ and $8z$ where the length of all data processed in a VM is $10z$. That means up to 80% of the processed data may be communicated between VMs.

To illustrate the difference between the bandwidth size and I/O data length, the former specifies required bandwidth allocation for each communicating VM pair while the latter specifies the size of the data that will be transferred on that allocated bandwidth. As the bandwidth size decreases and the data length increases, data transfer duration will extend. Another difference is in that the bandwidth size affects the subgraph search while I/O data length only affects VM's runtime performance.

Cloud locations are assumed to be user bases with varying **VM cluster demand** according to the human population density at that location. The demand is measured as the number of requests received from a user base. The most demanding user base requests 16 to 128 VM clusters of varying sizes during the simulation period of 50 hours. Arrival times of the requests are selected uniformly at random within the range of $[0,50)$ hours.

At each evaluation, only one of these four parameters varies while others are assigned their default values. Ranges and default values of simulation variables is given in Table 4.2. The variables in the experiments are defined as relative factors. For instance, in order to indicate that the VM requests generated during the simulations may contain $\frac{1}{64}$ to $\frac{1}{8}$ of a fixed cloud data center memory, VM memory range is indicated as $[x,8x]$

Table 4.2 : Ranges and default values of simulation variables.

Simulation Variable	Range	Default Value
VM memory size	$[x, 8x]$	$4x$
Inter-VM bandwidth size	$[y, 8y]$	$4y$
VM cluster demand per broker	$[16, 128]$	64
I/O data length	$[z, 8z]$	$4z$

and data center memory as $64x$. Same convention holds for other variables except VM cluster demand which is an absolute value. The simulation is run 30 times for each combination of variables. Upper limits of the ranges (i.e. $8x$, $8y$, $8z$ and 128) are selected for the following two reasons.

1. Higher values correspond to unrealistic cases such as a cloud data center that can only host a few VMs.
2. Simulations result in halt because overall cloud capacity runs short to answer demand.

Finally, the unit prices of resources are taken from Amazon Web Services. At the time of the evaluation, the price of a 50 Mbps AWS Direct Connect Port is \$0.03 per hour and the on-demand price of an EC2 instance with a 1 GB memory (t2.micro) is \$0.013 per hour. These rates increase in proportion to the port speed and memory size.

4.4.2 Baseline heuristics

TBM algorithm is evaluated against the following four VMCE heuristics that have separate node and link mapping stages. Heuristics mainly differ for their node mapping strategies while link mapping simply attempts to utilize the network connections on the shortest path between communicating VMs. Linear programming techniques which are widely discussed in the literature are not included since they are not feasible for the cases of more than a few nodes.

Random (RAN) Mapping Brokers submit VMs to random known available clouds.

Round-Robin (RBN) Mapping Each broker has an arbitrarily ordered list of known clouds and it probes them in a circular fashion.

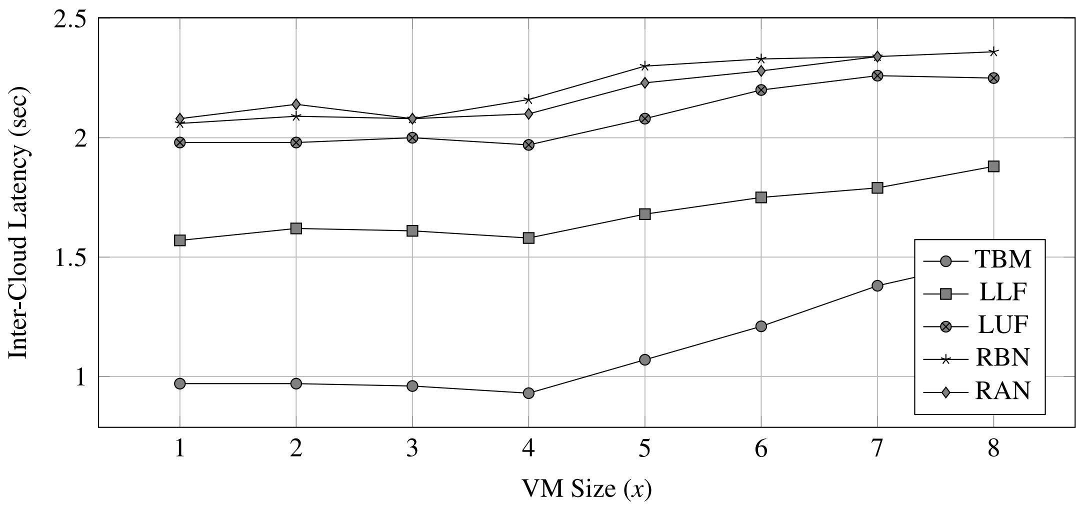


Figure 4.4 : Inter-cloud latency with varying VM size.

Least-Utilized-First (LUF) Mapping In order to exploit dynamic pricing strategy of the clouds and to balance their load, LUF mapping always submits a VM to the least utilized cloud which would have the lowest unit resource cost.

Least-Latency-First (LLF) Mapping LLF mapping is the same as RBN mapping except the list of clouds is sorted in ascending order of cloud-to-user latency instead of an arbitrary order. Assuming clouds that are close to the user in terms of latency would also be close to each other, objective is to reduce latency and increase VM performance.

4.4.3 Results and discussion

Selected results out of 32 performance criteria – evaluation parameter combinations are discussed in this section.

4.4.3.1 Inter-cloud latency

Figure 4.4 demonstrates the evaluation results with increasing size of VM. TBM has by far the least inter-cloud latency. Since RAN, RBN and LUF do not consider latency at all, they have the highest latency while LLF is around the middle ground. This indicates that inter-cloud latency assumption of LLF is correct to some extent, but not enough to reach TBM. As expected, latency increases as the VMs gets bigger. This is due to the increase in the rejection rate of the latency-optimized mappings.

An interesting result occurs when the variable parameter is the bandwidth (Figure 4.5). TBM performs around the same while the baseline methods seem to yield lower

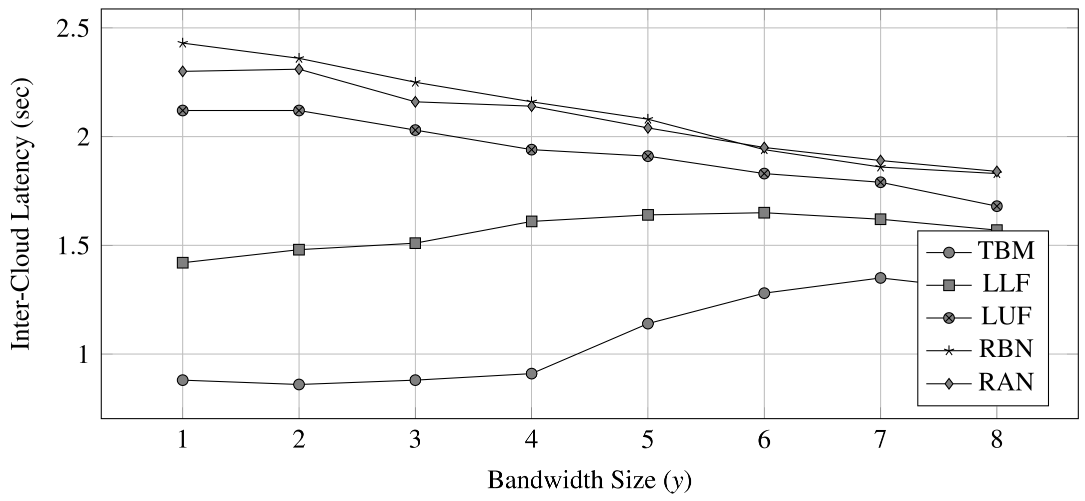


Figure 4.5 : Inter-cloud latency with varying bandwidth size.

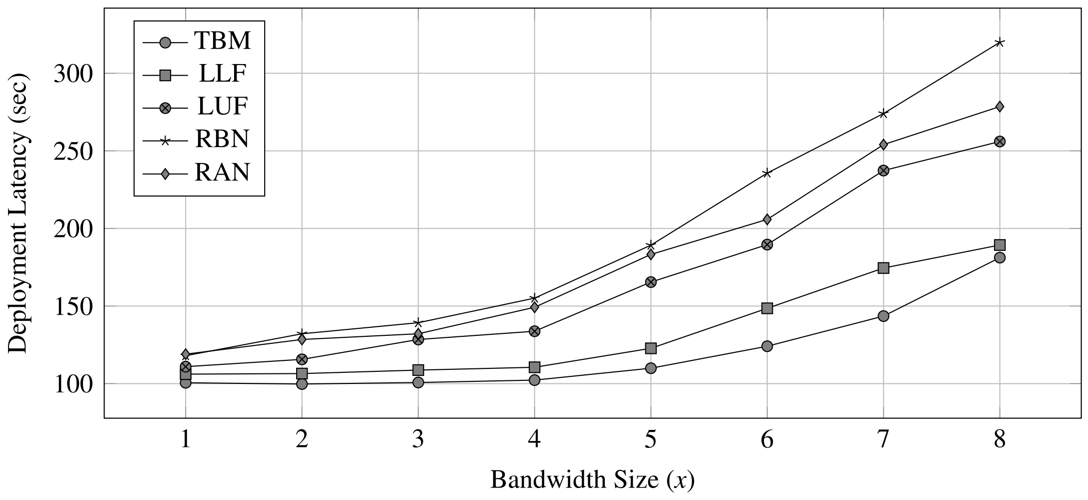


Figure 4.6 : Cloud-to-user latency with varying bandwidth size.

latencies as the bandwidth request increases. The reason behind this is the scarcity of bandwidth in the federation. When users request more bandwidth, utilization of network connections increases. As a result, link mappings with multiple hops are likely to be rejected. After several rounds of trial and error, even randomized heuristics come up with closely located VMs.

4.4.3.2 Cloud-to-user latency

All parameters yield similar results to the Figure 4.4 for the average cloud-to-user latency, hence only bandwidth size results are presented in Figure 4.6. Understandably, as the requests gets larger, possibility of deploying them close to their user decreases, thus latency increases. Although LLF makes decisions solely based on cloud-to-user latency, TBM performs slightly better even in average cloud-to-user latency criterion. This is due to the fact that LLF ignores efficient utilization of bandwidth capacity so

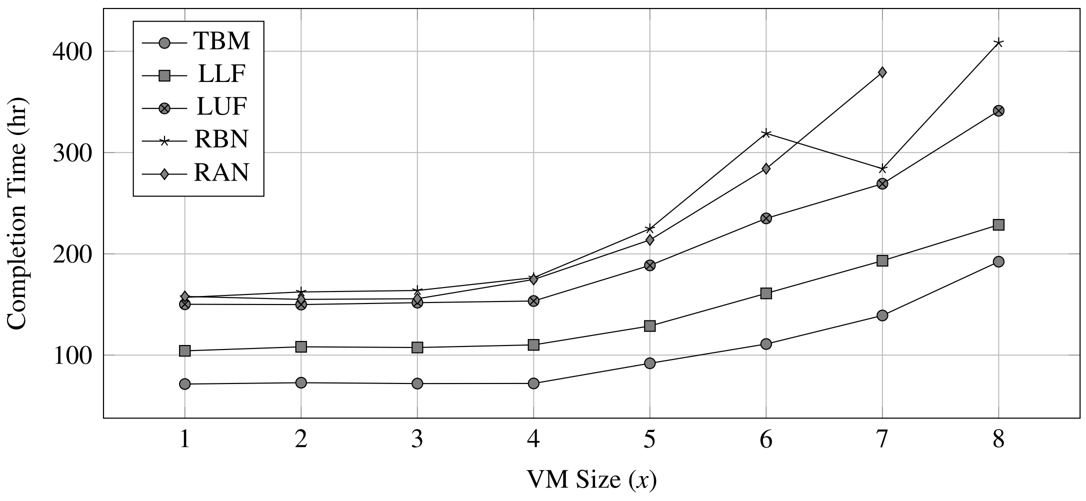


Figure 4.7 : Completion time with varying VM size.

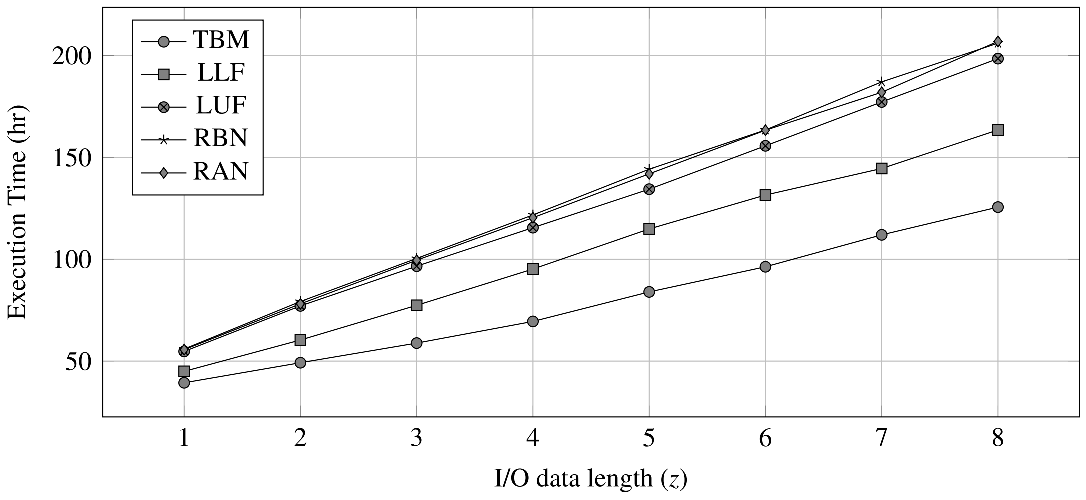


Figure 4.8 : Execution time with varying I/O data length.

the clouds with less latency cannot be utilized even when they have enough computing resources.

Cloud-to-user latency results indicate that, optimizing a single performance criterion and ignoring others in VMCE causes sub-optimal performance even for that criterion because the criteria are dependent to others. Multi-objective approaches such as TBM are more suitable for the problem.

4.4.3.3 Completion time and throughput

Impact of the TBM's latency reduction can be observed on the execution and completion times of the VM clusters as well as overall system throughput. Figure 4.7 show the average completion times according to the VM size. Execution time results with variable size are omitted as they are directly proportional to inter-cloud latency (Figure 4.4) while completion time also includes deployment latency and

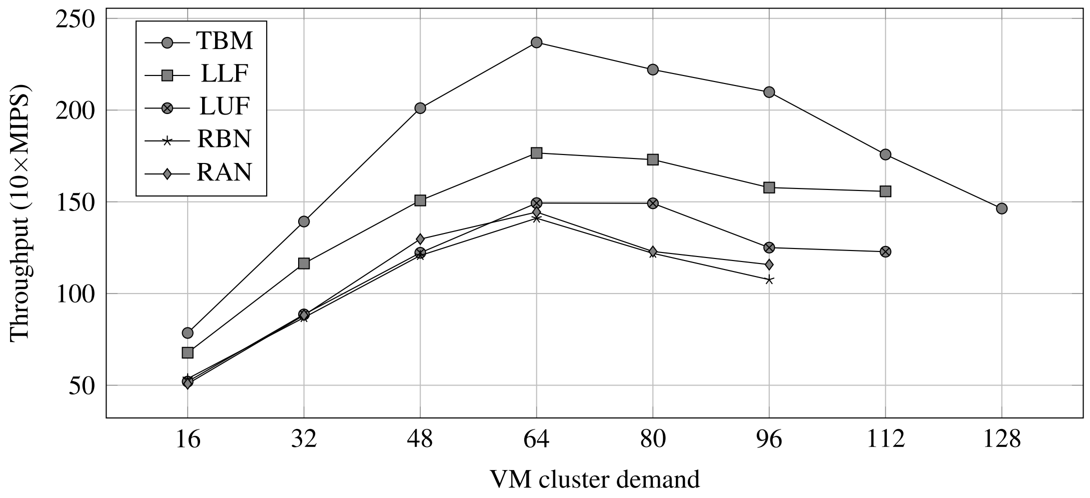


Figure 4.9 : Throughput with varying demand.

pending time. TBM reduces execution time up to 26% and deployment time up to 34% in comparison to the best performing heuristic (LLF). Additionally, Figure 4.8 demonstrates the average VM execution time with variable I/O data length. Again, TBM is clearly the best performer.

Overall throughput of the system increases until a threshold (around 64 in x axis) as demand increases (shown in Figure 4.9). However, brokers start to fail finding valid mappings and most new requests get rejected after that threshold. Although this is unlikely in a real life scenario, demand is kept increasing to test robustness of the algorithms. In that case, a deadlock may occur since some VMs of clusters are deployed and other VMs are waiting for resources utilized by the VMs of other clusters (a special case of dining philosophers problem). Missing data points and decreasing throughput in Figure 4.9 is due to such deadlocks. It is obvious that TBM algorithm can better utilize the resources of the system and it is more robust to demand peaks. A deadlock resolution technique is outside the scope of this thesis.

As seen in Figure 4.10, the rate of rejected VM gets higher for increasing load while lowest rate is achieved by TBM in all cases. Here, it should be noted that a rejection in RalloCloud is not permanent and simply means that the current available resources at the matched cloud does not allow to deploy the VM thus it should be dispatched to another provider. Likewise the throughput result discussed above, roughly the right half of the chart corresponds to unrealistically heavy workload which is useful to evaluate robustness of the algorithms. Under realistic workload, rejection rate of the TBM algorithm is under 30%.

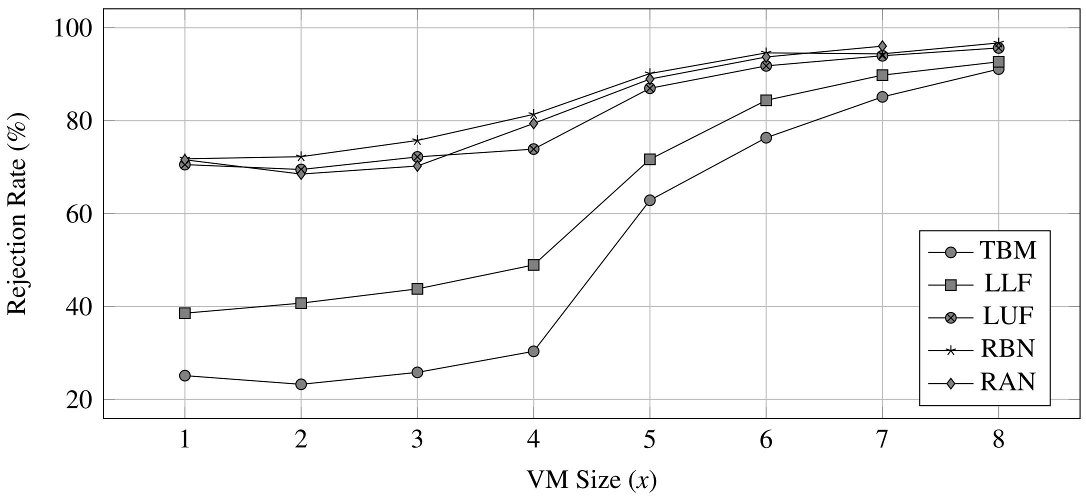


Figure 4.10 : Rejection rate with varying VM size.

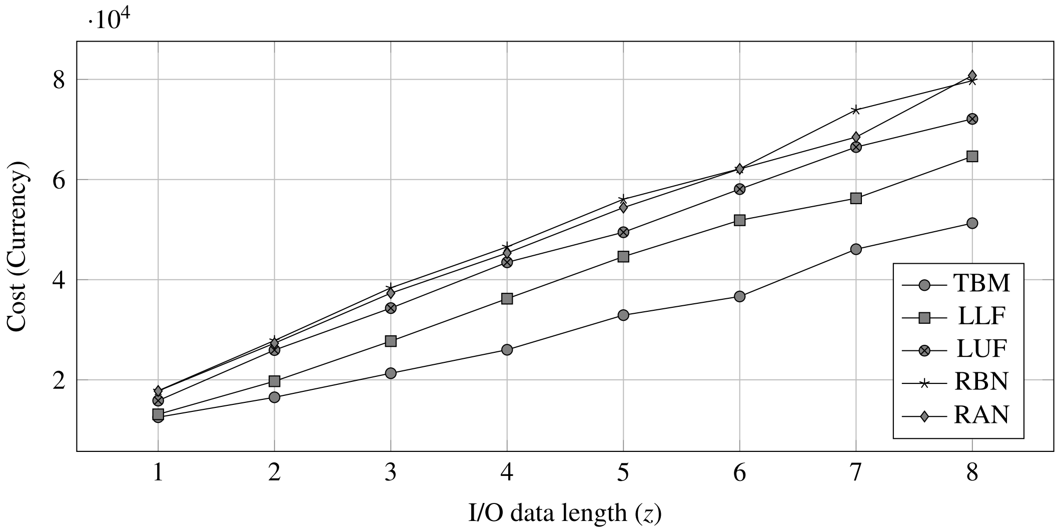


Figure 4.11 : Cost with varying I/O data length.

4.4.3.4 Cost

As explained in Section 3.1.5, total cost of a VM cluster is directly proportional to its execution time under the same unit price. Hence, change of costs in Figure 4.11 are quite similar to execution times in Figure 4.8. Although, LUF always maps VMs to the clouds with least unit costs, that does not result in lower overall cost than TBM or LLF. Because LUF does not take latency and bandwidth into consideration, its lengthy execution time becomes the determinant for cost. Similar to the case of suboptimal cloud-to-user latency performance of LLF, cost performance of LUF indicates that TBM is superior to heuristics that focus and optimize a single criterion due to its broad perspective of the whole aspects of the problem. It also shows that VMCE problem necessitates a multi-objective method to be solved effectively.

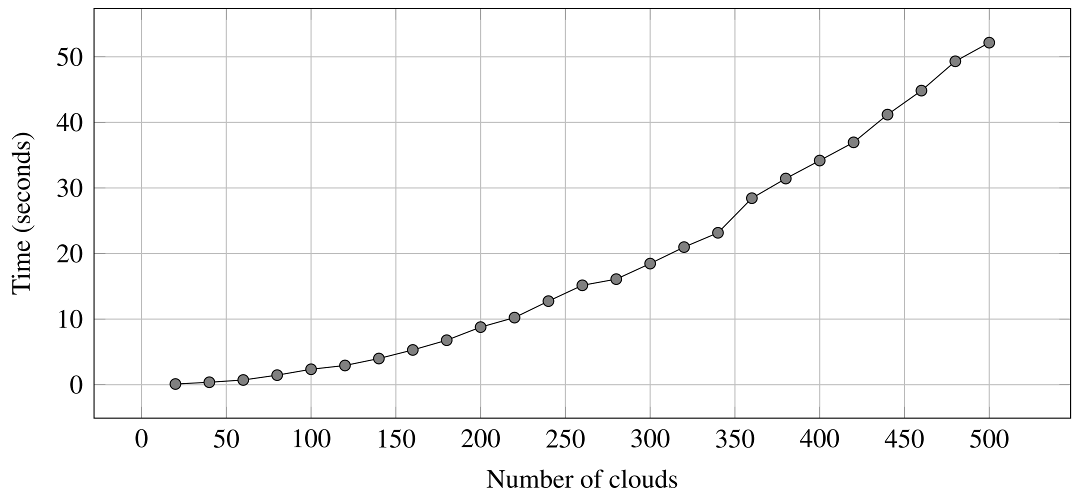


Figure 4.12 : Runtime performance with varying federation size.

4.4.3.5 Runtime performance

Finally, Figure 4.12 demonstrates the runtime performance of the TBM algorithm in larger federations. Four VM clusters with different topologies of five VMs are requested from each randomly generated federation. Federated cloud topologies are generated using the Watts—Strogatz model [108] with increasing number of vertices. Results are obtained on a workstation with Intel Xeon E5 CPU and 16 GB memory.

Results indicate that the algorithm can find a mapping and submit VMs to the corresponding clouds within a second for federations consisting of up to 70 clouds. Even in the extreme case of 500 clouds forming a federation, the algorithm can answer requests under a minute. Such a federation size can currently be quite unrealistic, however the experiments are conducted to identify the limits of the algorithm.

4.4.4 Resource Utilization

Utilization of resources within a single cloud data center are investigated to evaluate the algorithm and the evenness heuristic proposed in Section 4.3.3. The experiment aims to compare the power of heuristics to assign maximum number of VMs before a migration is necessary. Another aim is to assess the optimality of their placements. A good heuristic should place VMs in a way that migrations become compulsory as late and few as possible since migrations may cause temporary downtimes for VMs.

For each PM configuration, Table 4.3 contain average number of VMs assigned by 5 strategies, maximum number of VMs to fully utilize PMs and the rate of improvement

Table 4.3 : Average number of VMs assigned.

PM Capacity (x)	100	150	200	250	300	200	200	200	200	200
PM Count	8	8	8	8	8	4	6	8	10	12
RR	42,0	67,2	92,7	118,4	144,3	46,0	69,3	92,7	116,2	139,7
SP_{min}	46,2	73,2	100,2	127,2	154,3	48,7	74,4	100,2	126,2	152,3
SP_{dec}	46,0	72,6	99,3	126,0	153,2	48,4	73,8	99,3	124,9	150,7
SD_{min}	46,2	73,2	100,2	127,3	154,3	48,7	74,4	100,2	126,2	152,3
SD_{dec}	45,8	72,3	98,9	125,5	152,1	48,2	73,4	98,9	124,4	150,1
Maximum	53,3	80,0	106,7	133,3	160,0	53,3	80,0	106,7	133,3	160,0
Improvement	10,0%	8,9%	8,1%	7,5%	6,9%	6,1%	7,4%	8,1%	8,6%	9,0%

Table 4.4 : Average migration and perfect placement counts.

Strategy	Migration Count	Perfect Placement Count
RR	8,4	26
SD_{min}	5,5	108
SP_{min}	5,6	100

by the best performing strategy in comparison to the greedy round-robin baseline (*RR*). Subscripts *min* and *dec* stand for the minimum and most decreasing variants of the strategies, respectively. Decreasing variant can be obtained by calculating unevenness twice, i.e. before and after tentatively assigning the VM. In that case, actual assignment is made to the PM whose unevenness decreases the most, instead of the one whose unevenness becomes the minimum in the minimum variant.

It is evident that intelligent heuristics always perform better than the greedy assignment. In all 100 configurations, all strategies manage to assign more VMs than the round-robin. The worst improvement rate is $4,3\%$ (3 PMs with $300x$ capacity per resource) while the best one is $12,1\%$ (12 PMs with $75x$ capacity per resource). Utilization rate of round-robin is between $73,8\%$ and $90,2\%$ while utilization rate of the heuristics is between $80,5\%$ and $96,4\%$. Improvement rate gets better as the number of PMs increase (as seen in the right half of the final row in Table 4.3) but their capacities decrease (as seen left half of the same row). With high capacities and small number of PMs, randomness of resource requests tend to cover imperfect assignments while with low capacities and large number of PMs, decisions are more critical and should be made with more care.

Table 4.4 gives the average number of migrations introduced by the optimization after each three strategies (*RR* and two best performers of first experiment SP_{min} and SD_{min}) utilized the resources and stopped assigning more VMs. The table also contains the number of cases (among 1.000 executions) where heuristics made a perfect placement so that an optimization was not required. Results demonstrate that placements made by the heuristics are closer to optimum since the MIP optimization after the SD_{min} and SP_{min} strategies required respectively $34,5\%$ and $33,3\%$ less migrations than the MIP optimization after *RR*. They also made perfect placements in $10,8\%$ and $10,0\%$ of executions (MIP optimization doesn't migrate any VM) while the coincidental perfect placements are made by *RR* in only $2,6\%$ of cases. Intelligent heuristics make better placements that can be optimized with significantly less migrations. As a result, they allow VMs to run with less suspension as well as decreasing the need for an optimization algorithm. Span heuristic yielded roughly the same results as standard deviation and is preferable due to its simplicity.

5. REPLICATION MANAGEMENT

5.1 Problem Definition

The point of computation in cloud computing systems has begun spanning towards the terminal nodes of the network infrastructure in the last few years due to the availability of more powerful and smarter end devices. This expanse in the computational power triggered a diverse terminology including Fog Computing [109], Nano Data Centers [110], and Cloudlets [111]. Even though these concepts have their own differences and merits, their objective can be summarized roughly to disseminate tasks among a broader span of the distributed nodes in the cloud infrastructure instead of a small group of interconnected servers. Such approaches are referred as Edge Computing in this thesis.

Bringing the computation power to the edge of the network reduces latency and enables code offloading to the cloud. However, many services need to access data that is traditionally stored centrally. Thus, data access latency can be a bottleneck and override the benefits of Edge Computing especially for data-intensive services. In addition to the VM placement suggested in Chapter 4, continuous increase in the volume of data absorbed, circulated and processed in cloud systems also necessitate smart data distribution approaches. In this chapter, the focus is the dissemination of data in a distributed cloud computing system with a large number of nodes that are accessible for computational purposes. A decentralized approach to decide on the replication and placement of data originating from a central server towards the end devices in the cloud network is proposed. A cloud specific trade-off between the cost and latency is suggested as the main criteria in shaping an expansion parameter to determine the extent that the data is pushed towards the edge entities.

Client side caching is traditionally used in distributed systems to reduce data access latency. Similar to web caches, cloud caches are close to the clients and provide requested data locally in the case of cache hit or retrieve requested data remotely from

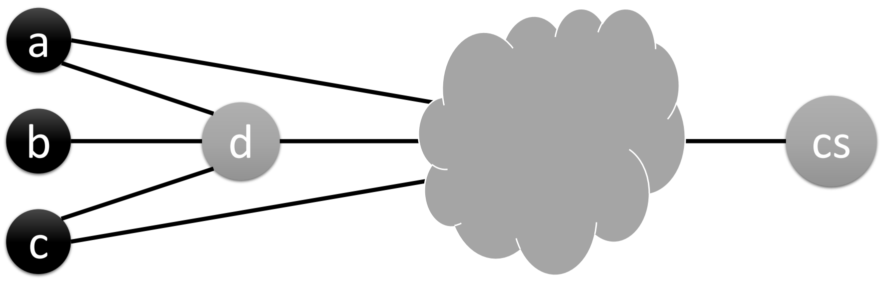


Figure 5.1 : An example Cloud topology to illustrate replication and caching [2].

the central storage in the case of cache miss [112]. However, caching results in low utilization of data copies since a cache can serve only the clients where it is stored. Replication, on the other hand, has the potential to provide a more cost-effective solution when the replicas are placed on critical network nodes and serve requests from multiple nearby locations. In that sense, replication can exploit the geographical locality of requests in addition to their temporal locality.

Figure 5.1 illustrates the problem with a trivial example. Assume that the edge entities *a*, *b*, and *c* are frequently requesting a data object. In the case of caching, all tree entities must store the data object to avoid fetching it from the central storage (*cs*) at each request with high latency. However, with smart replication, a single copy of the object can be placed on the storage node *d* which has low latency connection to *a*, *b*, and *c*, thus reducing cost and maintaining similar latency to local access.

Data replication is employed with several different objectives, e.g. increase availability, security, fault tolerance or reduce response time and bandwidth consumption. It is also effective in distributing the central storage load and improve scalability [61]. In this thesis, the focus is the performance benefits of replication with specific consideration of proximity of replicas to the clients to reduce latency and bandwidth consumption. The aim is to answer the following questions to minimize average replica-to-client distance in a bandwidth- and cost-effective way.

- Which data objects to replicate?
- When to create or destroy a replica?
- How many replicas for each object to create?
- Where to store each replica?
- How to redirect requests to closest replica?

Although replicating all objects to all storage nodes provides optimum proximity and latency, it is quite wasteful since the demand for each object varies and can be regional [113]. Moreover, in the cloud paradigm, a service provider (SaaS or PaaS) does not typically own the storage infrastructure but leases it from an IaaS provider on a pay-per-use basis. Thus, optimization of replica count and locations by considering the popularity of data objects is crucial to reduce cost [66]. Smart placement techniques can be especially effective in a Multi-Cloud scenario due to the availability of large number of geographically distributed storage options and the possibility to exploit pricing discrepancies across regions and providers [113].

Data accesses by geographically distributed users exhibit various patterns which can be summarized in three categories; temporal locality, spatial locality, and geographical locality [114]. Temporal and spatial locality are well-studied and addressed problems, however geographical locality, which is the focus of this thesis, recently gained more significance due to the increase in the magnitude of data being stored and processed in large-scale distributed systems. The locality classification is described below.

Temporal Locality A data object that is accessed by a user is likely to be accessed again by the same user. Caching systems exploit temporal locality to answer requests locally after the first request [115].

Spatial Locality A data object that is close or relevant to the previously accessed objects by a user is likely to be accessed by that user. When continuous blocks of data are stored instead of individual objects, caching can also make use of spatial locality with the assumption that the sequentially stored data objects are relevant to each other. There exists approaches (e.g. [116]) where relevant data objects are predicted using the past reference record and prefetched for local access.

Geographical Locality A common phenomenon in geographically distributed systems is that data objects that are accessed by a user is likely to be accessed by other nearby users. Although geographical locality is present in most distributed systems, for extreme cases one may consider a traffic jam where drivers in a certain area are demanding map/traffic data more intensely compared to a sparsely populated area. This kind of intensifying data demands can be very dynamic and hard to predict in practice. Another example might be a social event such as a sports

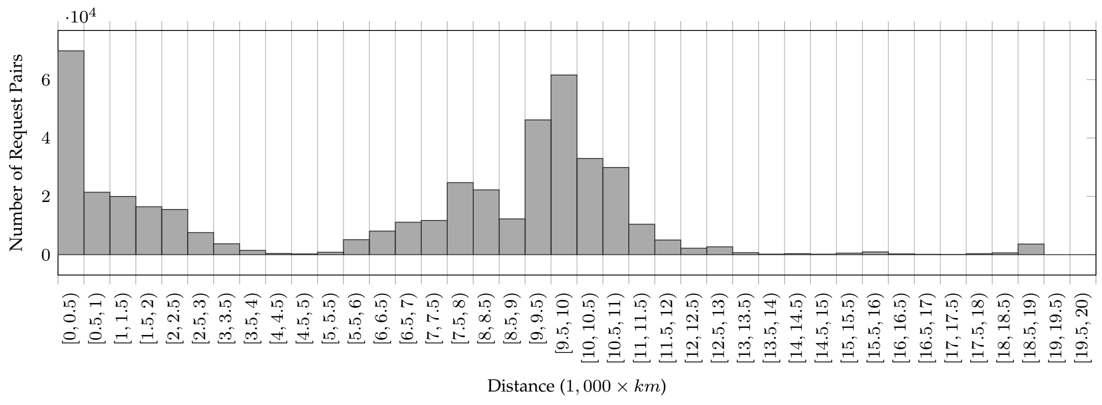


Figure 5.2 : Histogram of distances between data request pairs.

game or concert where the users continuously request similar data such as the video stream of a specific moment in the event. Geographical locality can also be almost permanent as it is in the scenario where mostly the residents of a town access its online public services or visitors access tourist guides.

The CAIDA Anonymized Internet Traces 2015 Dataset [117] is examined for a period of one-hour in order to demonstrate the extent of geographical and temporal locality in global Internet requests. A histogram of origin distances of requests for a single data object is shown in Figure 5.2. According to the analysis, more than 20% of the requests are originating from 1000 *km* distance to each other. When the diameter is increased to 2000 *km*, it covers nearly 30% of the requests. Similar results are obtained by others for the interactions between Facebook users in [94].

Caching cannot benefit from geographical locality because the users are unable to access the caches of other nearby users, in fact, they are unaware of them. Hence, a smart strategy for data replication and replica discovery is needed to reduce storage cost while maintaining similar or better data access time to caching.

5.2 Proposed Solution

Contribution of this thesis in terms of replica management is two-fold. First, a completely decentralized, dynamic, and online algorithm is proposed for the placement of data replicas across IaaS providers in an Edge Computing scenario. Second, a low-overhead messaging methodology is proposed to notify edge entities about nearby replicas so that they can submit their future data requests to them [2].

Replica Placement Algorithm Decentralized Replica Placement (D-ReP) algorithm is proposed where storage nodes that host replicas act as local optimizers by analyzing the experienced demand on the replicas. They evaluate the cost of storing replicas as well as expected latency improvement to make a migration or duplication decision to one of the neighbours. Their verdict may also be removing the local replica. Such decisions are made to maximize an objective function based on the Facility Location Problem (FLP) explained in subsections 2.1.2 and 3.1.3. The algorithm also allows user to control the trade-off between cost- and latency-optimization using an input parameter. Experimental results on real workload traces demonstrate significant improvements in replica access latency as well as network overhead and storage cost. Additional experiments on synthetically generated workload prove that the improvements are not limited to a single case.

Messaging Methodology Edge entities should be aware of the closest replica when they request a data object so that the promised benefits of the D-ReP algorithm are gained. However, complete awareness is only possible with centralized control or by broadcasting replica locations periodically. Here instead, a replica discovery approach is suggested where only the most relevant nodes are notified of the replica creations or removals.

5.2.1 Requirements and assumptions

The main requirement for the D-ReP algorithm is to place replicas across storage nodes (Cloud based storage providers and edge entities) in a way that the cost function given in equation 3.1.5 is minimized. There are also a number of nonfunctional requirements involved in the scenario.

First, the algorithm must be distributed and completely decentralized. As explained in Chapter 2, centralized algorithms are not suitable for the Edge computing scenario due to the large number of nodes and replicas. Collecting global topological and demand information in a centralized node and executing the complete optimization algorithm does not scale well with the node count [96, 97]. Communication between the nodes regarding the replica placement should also be kept to a minimum to avoid additional overhead. Second, Multi-Cloud and Edge Computing environments are highly dynamic. Edge users continuously enter and leave the network topology through connections with various latencies. Demand from each node and for each data object as well as the storage prices can also vary greatly over time. Thus, the number and locations of the replicas should be dynamic. Finally, the algorithm should be online since it is not possible to know future requests and environment.

Inputs to the D-ReP algorithm remain limited to the following items by taking abovementioned decentralization and low network overhead constraints into consideration. Each algorithm instance executing in a node is only aware of that node and its immediate neighbours in the network topology graph. It is assumed that the below listed information can be collected using standard monitoring tools (e.g. Skitter), DNS queries or routing tables.

- Number of requests for each replica that is stored in that node
- The neighbour node that each request is received through
- Perceived latency to each neighbour node
- Unit price of storage in each neighbour node

Another assumption is read-only data, similar to the other work on content distribution networks [64, 81, 114] so that data consistency is not an issue. Since Cloud and Edge Computing scenarios have the same data consistency requirements as traditional distributed systems and distributed data consistency is a well-studied area [118], it is left out of the scope. To apply the D-ReP algorithm to systems with frequent data updates, an independent consistency service can be utilized. In this case, any primary copy based distributed protocol can be implemented, e.g. Viewstamped Replication, Paxos or Zab. Most of other replica placement studies in the literature do not address the data consistency problem neither with similar justifications as here [59,64,79,81,83,91,110,114].

5.2.2 D-ReP algorithm

Two versions of the algorithm (i.e. source and edge) are triggered in equally spaced epochs and iteratively push replicas from the source node (central storage) towards the requesting edge nodes. A group of nodes in close proximity that repeatedly demand same data objects cause creation and migration of replicas towards them. The replicas are discarded or migrated to other locations when the demand fades.

The source version of the algorithm runs in the central storage node and can only create replicas of data objects in the neighbours of that node. The edge version, on the other hand, runs at each node where at least one replica is stored (i.e. in active node). A tiny VM for the execution of the algorithm is provisioned from the cloud provider when some storage space is leased for the first replica. The VM can turn itself off when no replicas are left or terminate after a period of inactivity. Edge version may decide to duplicate or migrate the replica to the neighbours, remove it, or do nothing. All decisions in both versions are the result of the comparison between the expected latency benefit and monetary cost. The epochs in different nodes does not need to be synchronized.

5.2.2.1 Source version

A replica of the data object k is created in the neighbour n of the central storage node c if the expected latency improvement is worth the storage cost of the replica. In other words, objective function in equation 3.1.5 should decrease after the replication. The following is the condition for the operation of creating a new replica.

$$num\_requests_{knc} \cdot latency_{nc} \cdot \lambda > unit\_price_n \cdot epoch \tag{5.1}$$

Here $num\_requests_{knc}$ is the total number of requests for k received through n by c in the most recent epoch(s). If the condition does not hold for any of the neighbours, no replica is created for that data object in the current iteration.

5.2.2.2 Edge version

A replica can duplicate itself to a neighbour n of its current host node h if the expected additional latency benefit of the new replica is greater than its storage cost. The condition below is the same as the new replica creation condition in the source version (Inequality 5.1) except c is replaced with h.

$$num\_requests_{knh} \cdot latency_{nh} \cdot \lambda > unit\_price_n \cdot epoch \tag{5.2}$$

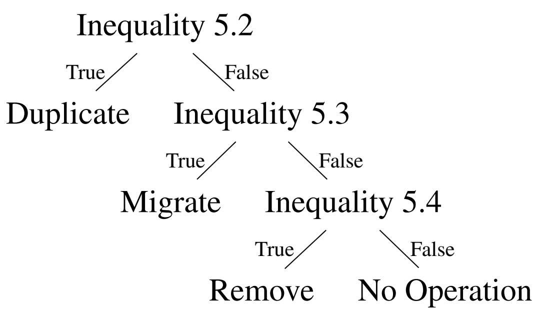


Figure 5.3 : Decision tree for the edge version of the D-ReP algorithm.

The migration condition in inequality 5.3 is evaluated if the duplication condition does not hold for a neighbour. Here, N is the set of all neighbours of h. The latency of the request received through n will decrease by the latency between n and h ($latency_{nh}$). Latency of the requests from all other neighbours is assumed to increase by the same amount since they will be answered through h. Actually, these neighbours may have lower latency connections to n and they may bypass h. However, the algorithm at h would not be aware of such connections since it has only local topology information for the sake of decentralization.

$$\Big(num\_reqs_{knh} - \sum_{\substack{i \in N \\ i \neq n}} num\_reqs_{kih}\Big) \cdot latency_{nh} \cdot \lambda > \big(unit\_price_n - unit\_price_h\big) \cdot epoch \tag{5.3}$$

A replica may get underutilized due to the creation of new replicas or changes in demand. In such cases, it should be discarded to avoid unnecessary storage cost. When the utilization of a replica in the last epoch drop below a certain percentage (α) of its expected utilization at creation, it is removed. The percentage, α, is defined in the range $(0,1]$ and chosen as 0.5 in the evaluation.

$$\sum_{i \in N} num\_requests_{kih} < original\_num\_requests_h \cdot \alpha \tag{5.4}$$

The decision tree that is evaluated at each active node for each replica and neighbour is presented in Figure 5.3. As the tree demonstrates, duplication operation has the highest precedence while removal has the lowest.

The user-provided parameter λ can be tweaked by the service provider to control the level of cost and latency. Greater λ values result in more aggressive expansion of replicas across the network, thus lower latency values in exchange for higher storage cost. The algorithm can also be extended to support more precise latency guarantees. To that end, the end-to-end latency of each request should also be collected and considered in creation, duplication, migration, and removal conditions.

5.2.3 Replica discovery

In replica-blind services, nodes that request data are unaware of the replica locations and they always submit their request to the central storage. If there exist a node with the replica of the requested data on the path, it answers the request [80]. Replica-blind services are typically implemented in domain-specific, single-tenant distributed systems such as CDNs. However, in a multi-tenant system such as the cloud, servers cannot analyze the requests flowing through them, hence replica-awareness is mandatory. Replica-aware services also make it possible to answer requests by the nearby replica locations that are not on the path to the storage.

A messaging system for replica discovery that complements the D-ReP algorithm is proposed. The messaging system is free of broadcast messages and central control in accordance with the decentralized design of the algorithm. The objective of messaging is to notify a node about a replica only if the node is expected to request it in the near future. The expectancy is inferred from both temporal and geographical locality of requests as explained below.

Each active node keeps a Known Replica Locations (KRL) table. The table stores the replica id, replica node, and latency to the replica node. It may contain several locations for each replica and is updated in the following occasions.

- When a replica is migrated or duplicated from n_1 to n_2, the new host n_2 notifies all nodes that requested the replica from n_1 in the most recent epoch(s) to exploit temporal locality. The list of such nodes is transferred from n_1 to n_2 together with the creation command and the replica data.
- n_2 also notifies its neighbours at both creation and removal of the replica to exploit geographical locality.

Latency to the replica node is approximated as the latency experienced in the notification message from n_2 and it is updated when a request is answered by n_2.

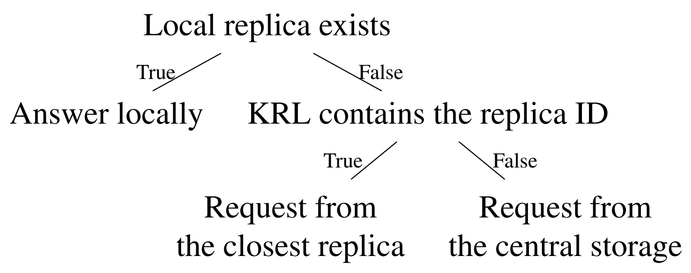

Local replica exists
True
False
Answer locally
KRL contains the replica ID
True
False
Request from the closest replica
Request from the central storage

Figure 5.4 : Decision tree to answer a data request at an edge node.

Figure 5.4 demonstrates the decision tree that is evaluated whenever a user request a data object at its local edge node. If a replica of the data is present at that node, then the request is answered with very low latency. If the replica is not stored locally but the node is aware of one or more replicas for that data, a request is submitted to the one with the least latency. Otherwise, it is requested from the central storage with the highest latency. Replicas with higher latency than the central storage are ignored.

5.3 Evaluation

5.3.1 Experimental setup

In order to evaluate the D-ReP algorithm and the baseline methods realistically, The CAIDA Anonymized Internet Traces 2015 Dataset [117] is used. The dataset contains anonymized passive traffic traces from CAIDA's 'equinix-chicago' high-speed monitor. Specifically, IPv4 packets data from February 19, 2015 between 13:00-14:00 (UTC) which contains more than 2.3 Billion records are used. In addition, GeoLite2 IP geolocation database[1] is used to map source IPv4 addresses to geographical locations. Moreover, the results are generalized using synthetic workloads based on uniform, exponential, normal, Chi-squared, and Pareto distributions of request locations in order to model various levels and types of geographical locality.

The network topology graph is generated using the BRITE tool [119] with the Barabási–Albert scale-free network generation model which is known to accurately represent human-made systems [120]. The topology contains 1000 nodes, 2994 edges and a heavy-tailed distribution of bandwidth in the range of 10 to 1024 mbps. For fairness, the location of the central storage in all three alternatives is selected as the node with the greatest closeness centrality in the topology graph. Amazon Web

[1]`http://www.maxmind.com/`

Table 5.1 : Ranges and default values of simulation variables.

Simulation Variable	Range	Default Value(s)
Epoch Length (min)	$[1,20]$	$3,10$
λ	$[0.01,0.20]$	$0.10,0.16$
Cache Capacity	$[10,200]$	30

Services S3 prices[2] are chosen to calculate the storage and transfer costs for the data objects and replicas.

For the caching system which is implemented as a baseline, the number of replicas that can be stored at each cache is limited. When the cache capacity is full, least recently used (LRU) cache replacement strategy is used. Simulation variables are the epoch length, λ and cache capacity. The ranges and default values of variables are given in Table 5.1. For each experiment, one of these varies while the others are assigned their default values. The range limits are chosen by ensuring the best performance of the D-ReP and caching algorithms as well as considering practical constraints. Specifically, unrealistically short and long epochs, or too small cache capacities are not used even if the algorithms perform well, since they would have no real-world application.

5.3.2 Evaluation with real workload traces

5.3.2.1 Latency and cost

Latency improvement rate is used to measure the extent that a replication or caching solution decreases average access latency to replicas with respect to no-replication solution. In Figure 5.5, latency improvement of the algorithm with varying λ values is presented. Since λ is only effective for D-ReP, latency improvement of caching is fixed at 25.60%. As the λ value increases, D-ReP gets able to afford more replicas especially at outer locations. Hence, the average latency to the replicas decreases steadily to the levels comparable to caching and beyond.

Increasing epoch duration (e), as shown in Figure 5.6, also improves latency to some extent. However, the increase is not as steady and converges after around the epoch duration of 7 minutes. Longer epochs have the advantage of aggregating more data

[2]`https://aws.amazon.com/s3/pricing/`

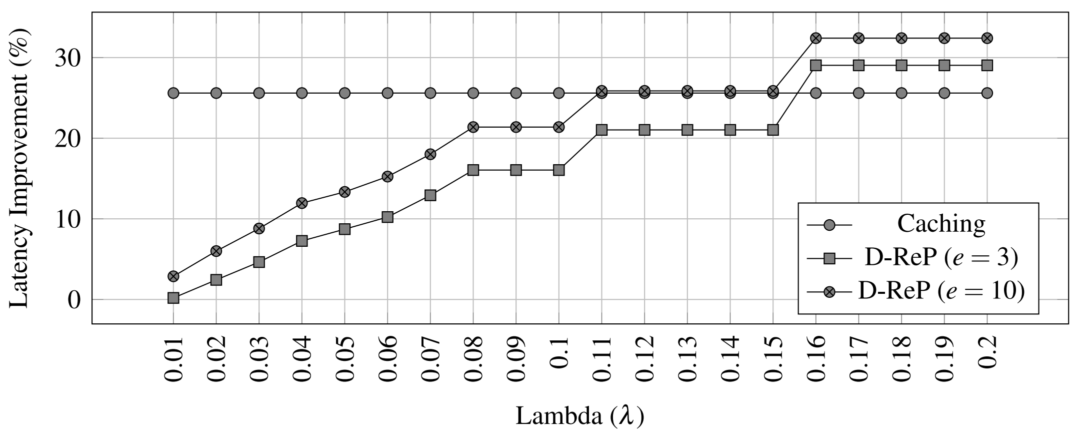


Figure 5.5 : Latency improvement rate with variable λ.

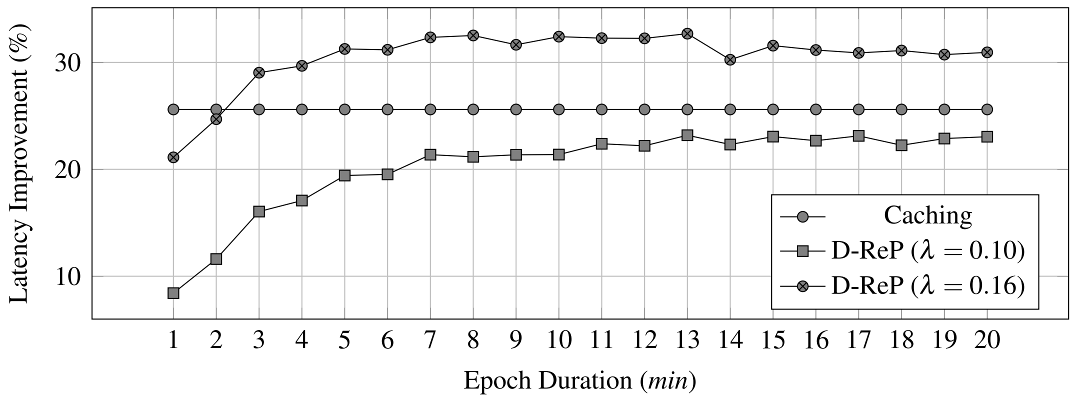


Figure 5.6 : Latency improvement rate with variable epoch duration.

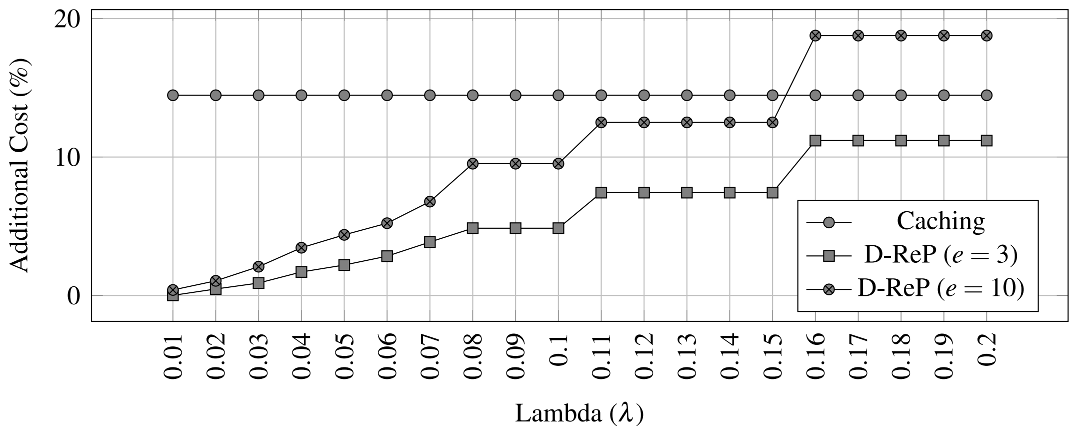


Figure 5.7 : Cost increase rate with variable λ.

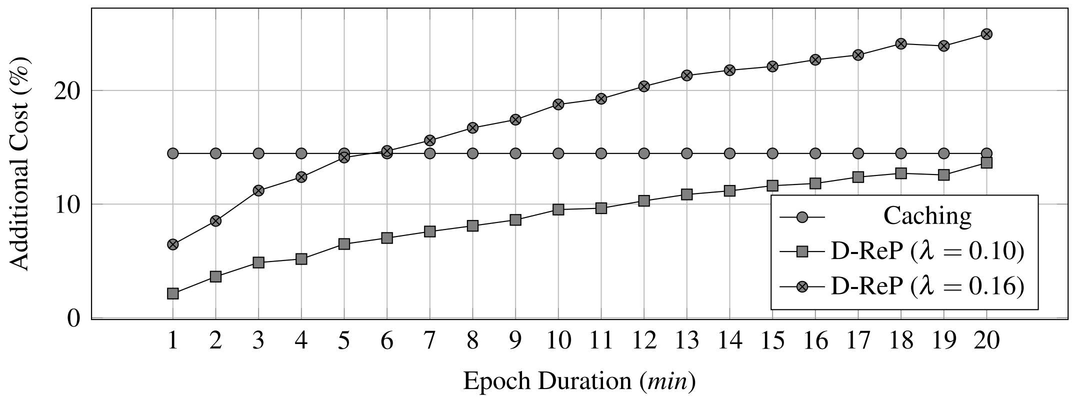


Figure 5.8 : Cost increase rate with variable epoch duration.

to make sense in replication decision, but they harm the reactivity of the algorithm to relatively small changes in demand.

Another measurement is the rate of increase in replica storage cost that each method causes on top of the central storage cost. Figures 5.7 and 5.8 clearly demonstrate that D-ReP incurs significantly lower cost than caching in all cases expect very high λ values (≥ 0.16) and very long epoch durations (≥ 15 *mins*).

Absolute cost and latency values of the no-replication solution are required to better interpret the relative improvement rates in Figures 5.5 to 5.8. Average data access latency is $1.82s$, while data storage and transfer cost per request is \$0.24. Hence, a latency improvement rate of 30% corresponds to a $0.55s$ gain in time which is quite significant as the number of data objects requested by a user in a single session can easily reach tens if not hundreds.

5.3.2.2 Benefit-cost ratio

Benefit-Cost Ratio (BCR) indicator summarizes the overall value for money of a proposal and is useful to decide between options when the most profitable is not obvious. Greater BCR values are favourable and generally, proposals with a BCR less than 1 are rejected. Here, it is used to evaluate the efficiency of algorithms to address the trade-off between data access latency and data storage/transfer cost. and calculated as latency improvement rate divided by cost increase rate (as shown in equation 5.5).

$$BCR = \frac{Rate\ of\ Latency\ Improvement}{Rate\ of\ Cost\ Increase} \tag{5.5}$$

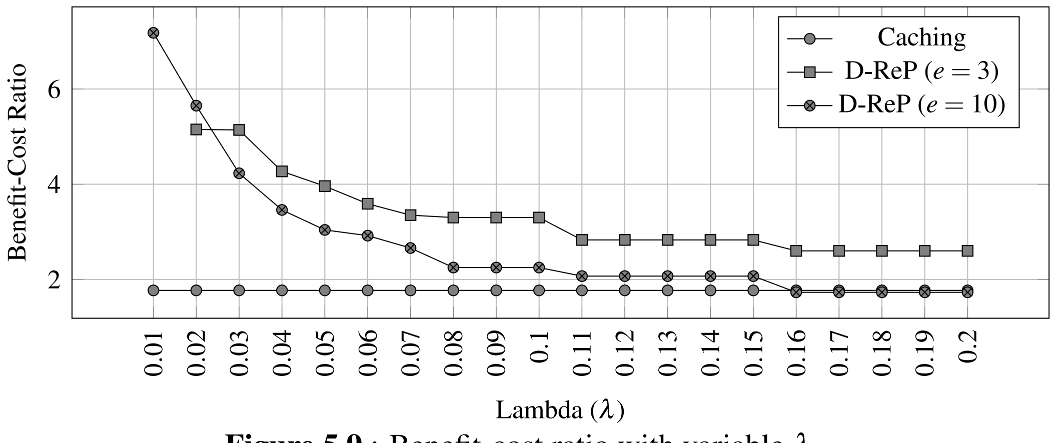


Figure 5.9 : Benefit-cost ratio with variable λ.

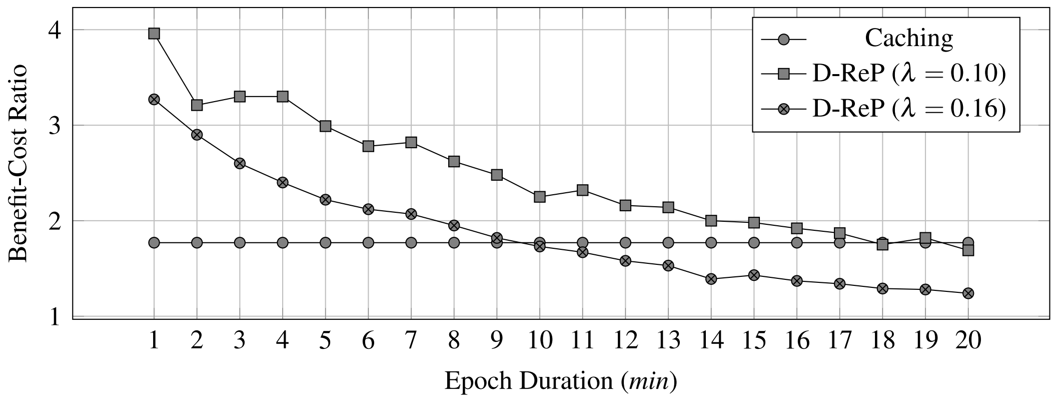


Figure 5.10 : Benefit-cost ratio with variable epoch duration.

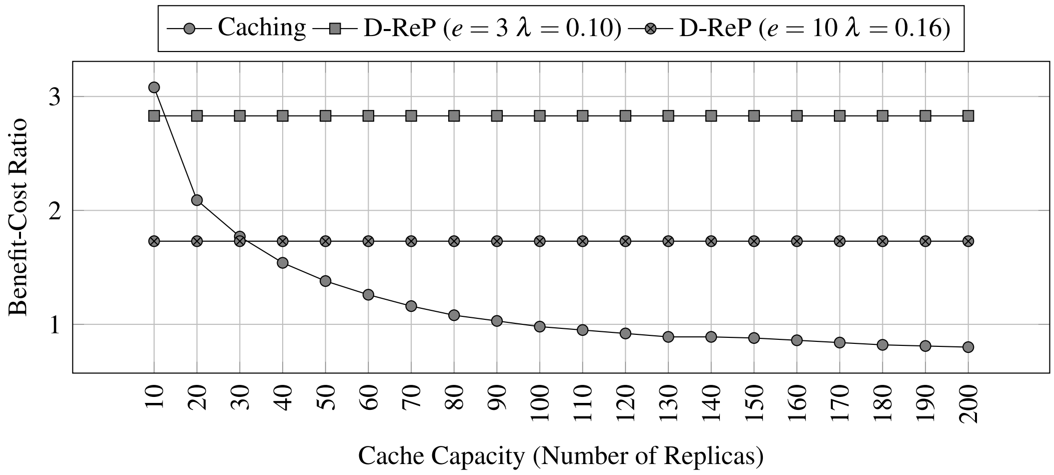


Figure 5.11 : Benefit-cost ratio with variable cache capacity.

D-ReP is more cost-efficient in all λ values, as shown in Figure 5.9, with 1.93 times greater BCR than caching on average. Considering this result with Figures 5.5 and 5.7 reveals that λ can be used to control the desired level of latency in a cost-efficient way. First data point of 'D-ReP ($e = 3$)' is not displayed in this chart as the BCR value is unrealistically high (18.87) because the denominator (additional cost) approaches zero. Figure 5.10, on the other hand, shows that longer epoch durations can be less efficient than caching. Although, latency improvement converges in Figure 5.6, cost continues to increase steeply in Figure 5.8. Thus, epoch duration does not come out as an appropriate way of controlling latency.

Although, a fixed BCR is presented for caching in Figures 5.9 and 5.10, efficiency of caching also varies by cache capacity. Larger capacity means higher possibility of cache hits and thus lower average latency. Figure 5.11, shows that the BCR of caching is significantly lower than D-ReP in most cases. It is only comparable in very small cache capacities where the latency improvement is limited. These results indicate that caching can only be effective to reduce latency in a small amount with very low cost.

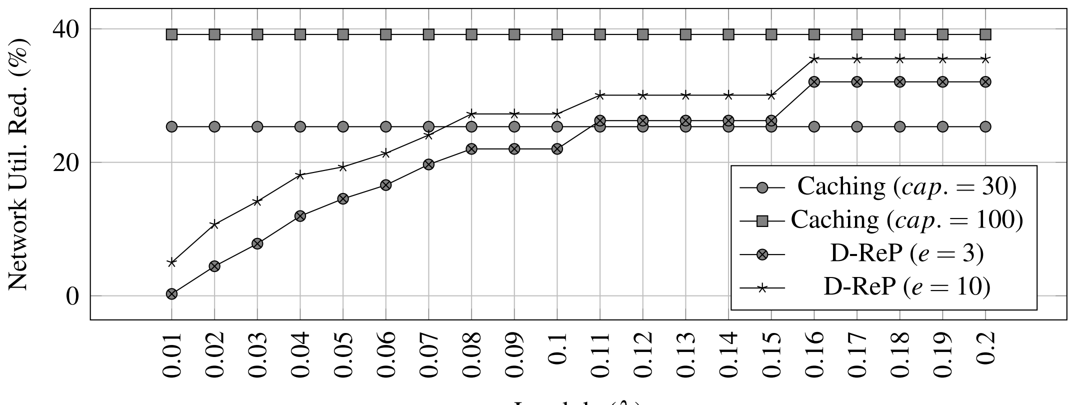


Figure 5.12 : Network utilization reduction with variable λ.

It is an expensive and inefficient method to achieve significant latency improvements (e.g. more than 20%).

5.3.2.3 Network overhead

Both D-ReP algorithm and caching also reduce overall network utilization since some requests are answered from nearby locations and utilize less connections. Figure 5.12 demonstrates the reduction in network utilization relative to the single storage (no-replication) solution. Results are comparable for most λ values.

Being a replica-aware solution, D-ReP algorithm requires a replica discovery strategy as described in Section 5.2.3. Notification messages for replica discovery cause a network overhead in addition to the regular data requests and responses. A strategy that avoids broadcasting and central control of replica locations is proposed so that the network overhead is minimized. However, replica discovery is not optimal due to this constraint. That is, some nodes may send their requests to replica locations that are erroneously in their KRL table but actually have been removed. This synchronization failure may also incur a network overhead and deteriorate the performance of the D-ReP algorithm. Table 5.2 presents the percentage of notification messages among all messages, percentage of failed requests among all requests, and percentage of latency caused by these failed requests. The results indicate that all of these are negligible.

Table 5.2 : Percentages of network overhead factors.

Percentage of Notification Messages	2.63%
Percentage of Failed Requests	819 *ppm*
Percentage of Latency Due to Failure	$1,108$ *ppm*

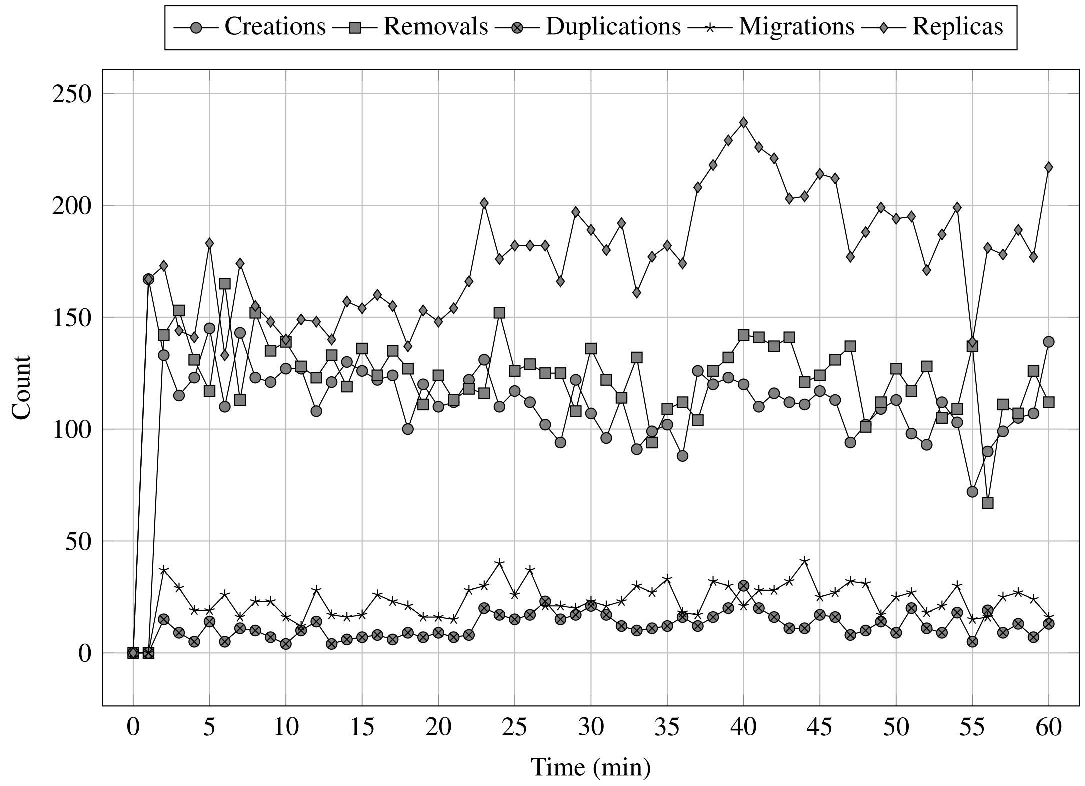


Figure 5.13 : Number of replicas and completed operations in time.

5.3.2.4 Convergence

The change of the number of replicas present in the system in time is also evaluated in addition to the performance indicators. Figure 5.13 presents the replica count in one-minute intervals as well as the occurrence of each of four operations. The results show that the number of creations and removals are more or less the same and the replica count converges around 200 replicas. Although, the number of migration and duplication operations are relatively smaller than creations, these are the main factors of the D-ReP algorithm that allow replicas to move closer to the requesters.

5.3.3 Evaluation with synthetic data

With the purpose of generalizing the results and conclusions in Section 5.3.2 using the CAIDA Internet Traces, synthetic user demands are also generated. Various probability distributions are used to determine the locations of the request so that various levels and forms of geographical locality are experimented. Using a random distribution, each data request is mapped to a 1000×1000 grid, same size as the network topology explained in Section 5.3.1. Then, the requests are mapped to the network node that is closest to them.

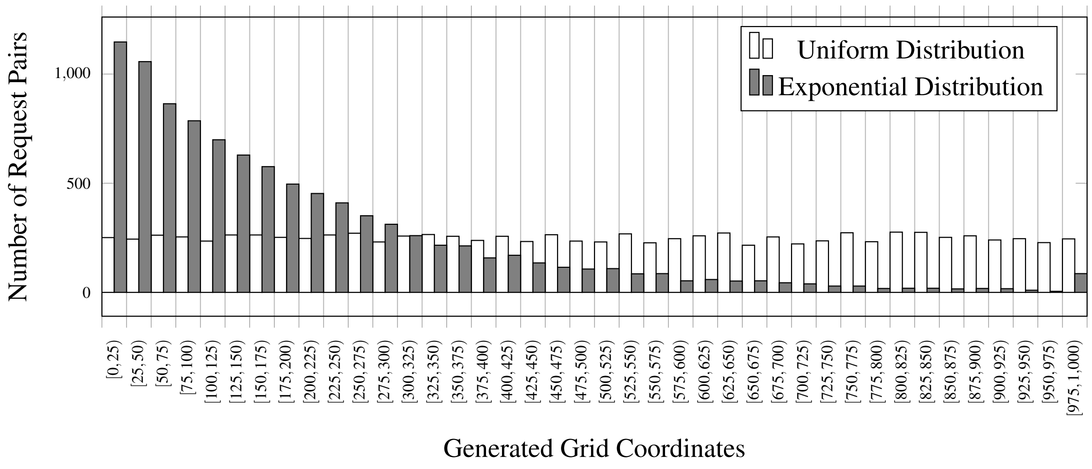


Figure 5.14 : Data request locations with uniform and exponential distributions.

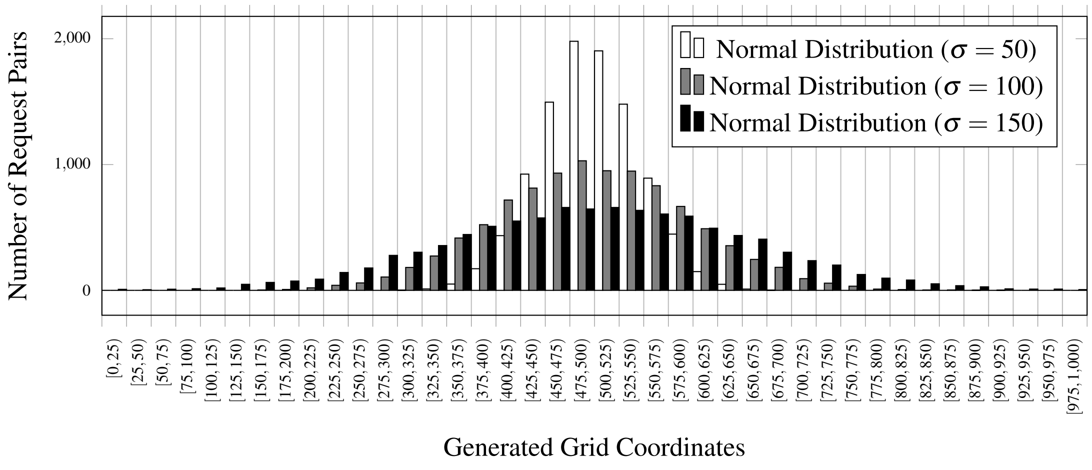


Figure 5.15 : Data request locations with normal distribution.

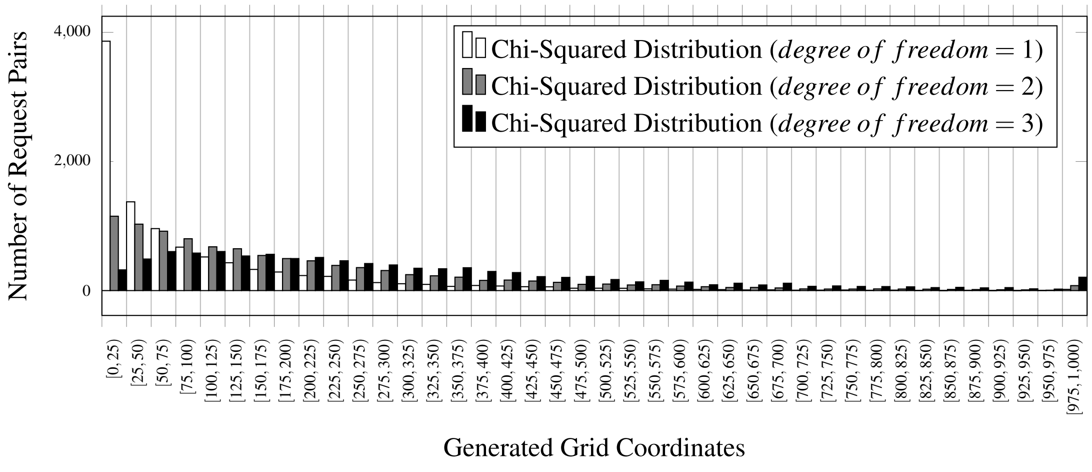


Figure 5.16 : Data request locations with Chi-squared distribution.

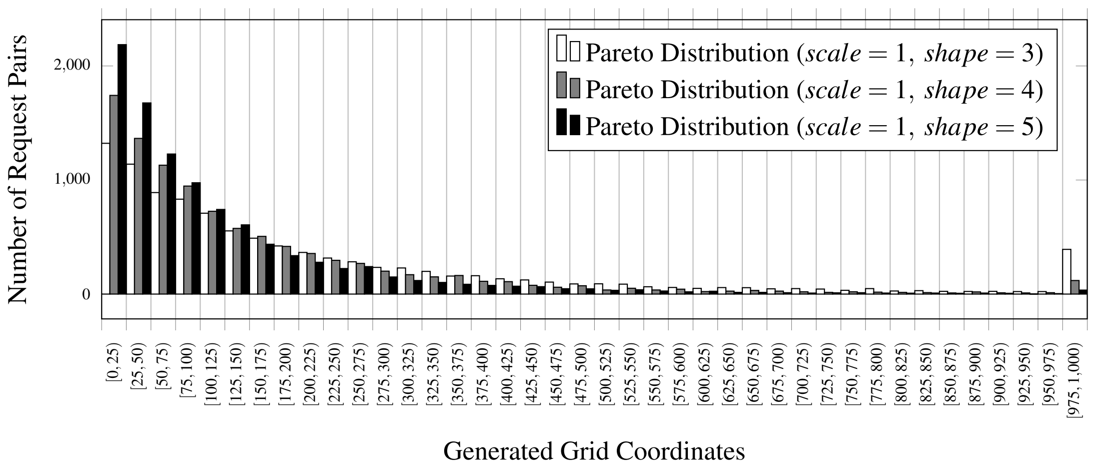


Figure 5.17 : Data request locations with Pareto distribution.

5.3.3.1 Probability distributions of the request locations

D-ReP algorithm is simulated in several cases in which data request locations follow a uniform (Figure 5.14), exponential (Figure 5.14), normal (Figure 5.15), Chi-squared (Figure 5.16), and Pareto distribution (Figure 5.17). For the last 3 distributions multiple parameter values are also used to reach a total 11 different distributions. Note that, although the Figures 5.14 to 5.17 demonstrate one-dimensional distributions, multivariate versions are used in the experiments for the location in two axes. The number of requests is $100,000$ for each distribution.

5.3.3.2 Results and discussion

The results in Figure 5.18 demonstrate that D-ReP yields latency improvements in each and every case. However, the rate of improvement depends on the geographical distribution of the requests. Normal distribution with a standard deviation of 50 produces the best latency improvement while uniform distribution is the worst. BCR results in Figure 5.19 are also similar in terms of relative performance order of the distributions.

The results are not unexpected because uniform distribution induces no geographical locality in data requests. D-ReP specifically makes use of geographical locality and thus has little or no impact for uniform distribution. However, as the non-uniformity increases, outcome improves. To demonstrate the effect of uniformity, Figure 5.20 is presented where the variance of the probability density function (PDF) of a distribution is mapped to the BCR achieved with that distribution. There is a strong correlation

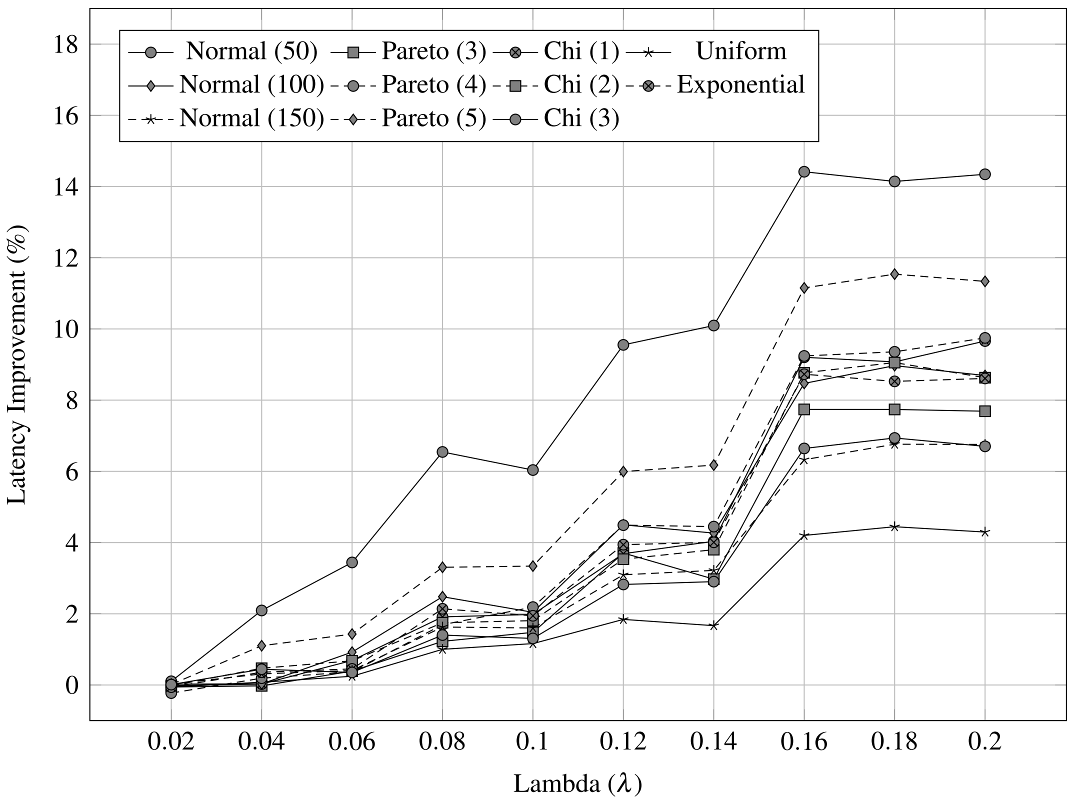


Figure 5.18 : Latency improvement rate with variable λ.

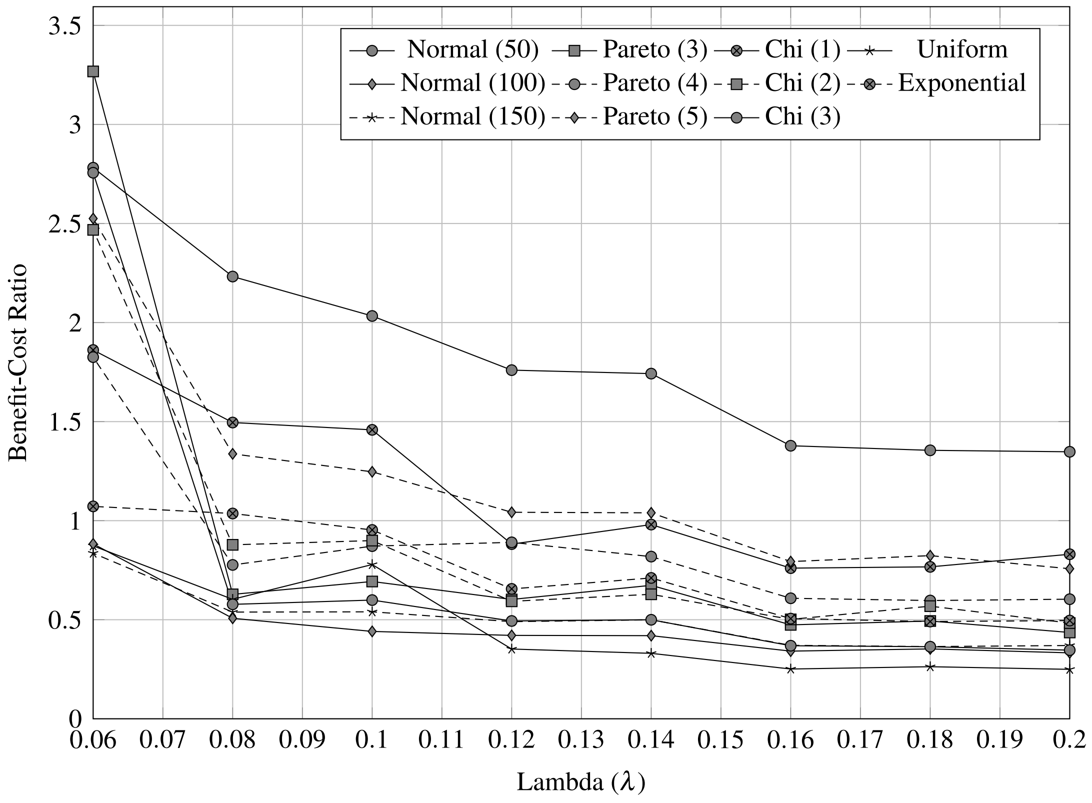


Figure 5.19 : Benefit-cost ratio with variable λ.

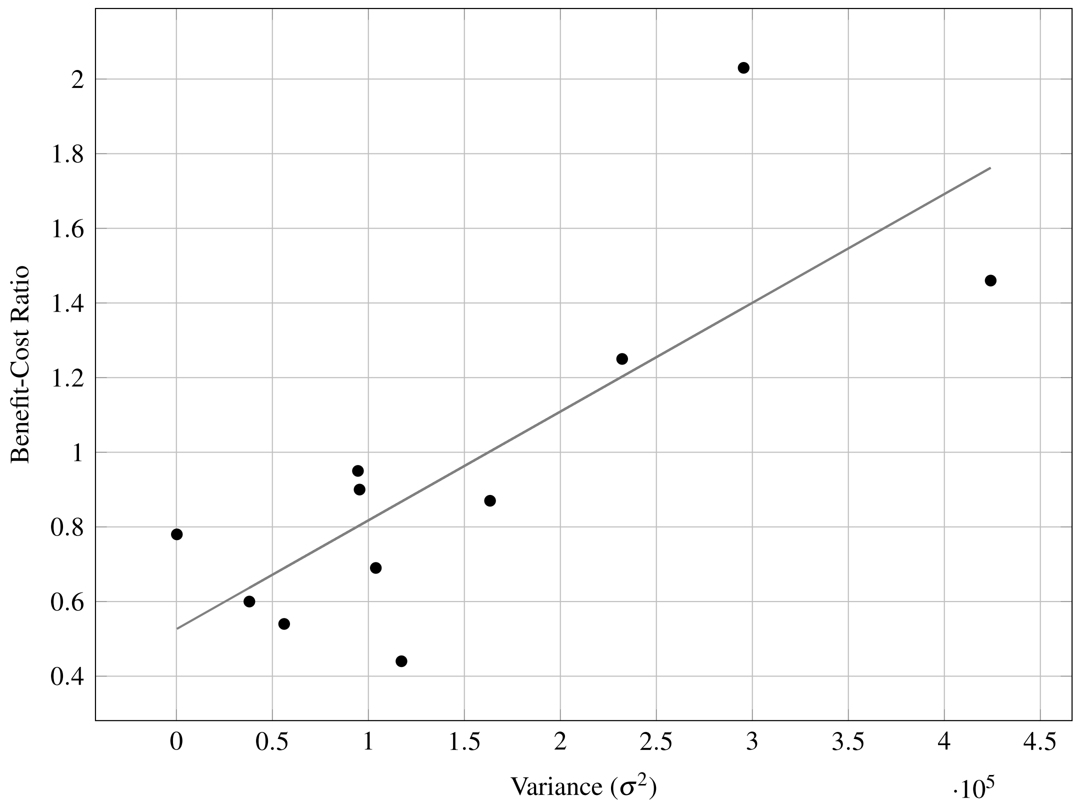


Figure 5.20 : Variance of the distributions and the BCR ($\lambda = 0.1$).

between these two variables with a coefficient of 0.78. It can be concluded from these results that D-ReP algorithm is most effective in cases where the PDF variance of the location distribution is high, or in other words, geographical locations of request are densely clustered in certain areas. This is usually the case in real workloads.

6. CONCLUSIONS AND RECOMMENDATIONS

Cloud needs to transform into a completely decentralized, federated and ubiquitous environment similar to the historical transformation of the Internet to be able to handle the changes in its usage scenario. During this transformation, distributed management of resources emerges as a significant obstacle. This includes the discovery, allocation, and monitoring of resources. In this thesis, optimized management of limited computing and network resources to adapt to the decentralization is proposed. Specifically, cloud services that consist of several virtual machines, dedicated network connections and databases are mapped to a multi-provider, geographically distributed and dynamic cloud infrastructure. The objective of the resource mapping is to improve QoS in a cost-effective way. To that end; network latency and bandwidth as well as the cost of storage and computation are subjected to a multi-objective optimization.

6.1 Contribution

Resource mapping is carried out in two phases. In the first phase, VMs and their inter-dependencies are mapped to the distributed cloud infrastructure. In the second phase, the data objects in the storage layer are replicated and mapped to the same infrastructure. These two phases, complement each other in the reduction of storage and processing costs as well as the improvement of QoS. Detailed experiments under real-world as well as synthetic workloads prove that the hypotheses of the both phases are true. The hypotheses given in section 1.2 were as follows.

1. Mapping the VMs of a cloud service to a subset of data centers that have the similar network topology as these VMs would outperform greedy methods.
2. Mapping the data replicas of a cloud services by considering the level and the location of their demand would outperform data caching.

Main contributions in the thesis are (1) realistic modeling and simulation of the resource mapping in the Inter-Cloud environment, (2) TBM and Minimum Span algorithms for VM mapping, and (3) D-ReP algorithm and KRL technique for data replication and mapping.

6.1.1 On problem modeling

Resource modeling and allocation problems get more complicated when a distributed scenario such as inter-cloud is considered instead of a single data center. Inter-cloud (or Cloud Federation) is a distributed model for cloud computing where coordinated cloud providers share their resources and dispatch workload to each other. The most significant benefits of the model to the provider are better scalability, geographical coverage, and resilience.

Resource mapping problem is modeled as a concrete framework where interworking of cloud providers and virtual machines as well as scaling workload across multiple clouds are supported. In addition to classical resource allocation parameters, the framework also models network related issues such as latency and bandwidth constraints, thus provides easy migration, high QoS for geo-distributed demand, and the possibility to exploit vendor pricing policies. It also allows realistic simulation of the resource mapping algorithms and provides useful criteria to compare them. Proposed framework is the first simulator for distributed cloud systems including multi-cloud, federated cloud and edge computing.

6.1.2 On virtual machine placement

Network topology should be taken into consideration in order to realize the abovementioned benefits of inter-cloud. Thus, a novel virtual machine cluster embedding algorithm called Topology Based Mapping (TBM) is proposed. It aims to find an efficient mapping between the physical Inter-Cloud topology and user demands in the form of virtual topologies. Consequently, performance degradation due to network latency is minimized and doesn't override the benefits of multi-cloud deployment. TBM employs a graph theoretical approach (i.e., subgraph isomorphism) in combination with greedy heuristics to achieve its goals.

The main objectives of the algorithm are to reduce network delay and optimize bandwidth utilization. Comprehensive evaluation demonstrated the efficiency of the resulting resource allocation as it achieved better job execution time (makespan), throughput, rejection rate, average network delay and average resource cost in comparison to the outputs of the baseline methods under various experimental configurations. Experimental evaluation prove the hypothesis that the structural information about a cloud service is beneficial for an effective resource mapping which improves QoS at low cost. Specifically, topology based mapping can decrease the latency between VMs by 64%, completion time by 54%, and cost by 36% in comparison to greedy algorithms. It can also increase the system throughput by 70%. TBM algorithm is the first attempt to map virtual topologies to their isomorphic subgraphs in the physical Cloud or Grid topologies.

6.1.3 On replica placement

As the volume and velocity of data in the cloud is increasing, the geographical distribution of where it is produced, processed and consumed is also gaining more significance. It is getting less feasible to move data to a distant data center for processing and move output again to the consumer location. Several promising approaches including Cloudlets and Edge Computing are instead suggesting to bring processing entities to the edge of the cloud network to reduce latency. This is especially useful in code offloading for mobile cloud applications.

One issue in this scenario regarding resource management is the latency between the processing entity and the data. Although the above-mentioned approaches reduce the latency between the user and the processing entity, the data required for the cloud application is usually stored in a centralized provider. It is not feasible to replicate entire data in large number of geo-distributed locations due to economical factors. In addition, edge nodes (e.g., cloudlets, nano data centers) have limited storage capacity in comparison to cloud infrastructure. Hence replication of individual data objects on multiple locations based on the magnitude and location of user demand as well as storage pricing is proposed in attempt to reduce data access latency.

Optimal selection of the number and location of the replicas is a challenging problem due to the varying/mobile nature of user demand and the trade-off between cost (number of caches) and latency. Moreover, knowledge of the complete topology including capacities, latencies and prices in such a fine-granular infrastructure is not realistic. Thus, a distributed and context-aware replica management is suggested.

Decentralized replica placement can either yield the same latency improvement with 14% less additional cost than caching or improve latency by 26% more with the same additional cost, depending on the chosen value of the trade-off control parameter. It is the first completely decentralized replica placement algorithm that can work with only partial local knowledge.

6.2 Future Work and Limitations

Distributed and federated clouds are still in their infancy. Significant future work is required to realize the infinite scalability promise of the cloud computing. Moreover, resource mapping framework suggested in this thesis is also open to improvement. The following list of topics are identified to recommend future research directions.

- Suggested resource mapping framework would fully serve its purpose when the unification of geographically distributed cloud providers becomes prevalent. The most important issue regarding the unification is the standardization of the application programming interfaces by IaaS providers. This standardization will not only allow Federated Clouds but will also leverage Multi-Clouds with simpler broker implementations.
- Both topology mapping and replication management algorithms provide best-effort latency improvement and cost reduction. Although they outperform existing alternatives, certain cloud services may require real-time performance and cost guaranties. Hence the algorithms can be extended with such capabilities.
- Another area that is open to improvement is the self-awareness of cloud services. Services should be able to monitor their tasks in IPs, translate low-level metrics to QoS metrics, and proactively act to increase their performance and limit their spending. Actions may include migration as well as horizontal or vertical scaling.
- Proposed resource mapping framework does not pay regard to load balance and fairness across cloud providers. These considerations may be required for the business logic of the cloud unification.
- Fault tolerance and availability are not considered. Although replication already provides such benefits, they can be formally modelled and evaluated.

- Read-only or rarely changing data is assumed so data consistency is left out of the scope. To extend the algorithms for dynamic data sources, an external primary copy based data consistency mechanism can be implemented.

6.3 Implementation Issues

In practical implementation of the proposed algorithms, host locations and motivation of the involved parties should be taken into consideration. One scenario can be that a PaaS provider can act as a matchmaker between the user and the cloud vendors. In this case, the PaaS provider should provision small-size VMs in all included data centers and edge nodes to execute the resource mapping algorithms. In return, the provider can charge the user not more than the expected cost benefit of the algorithms. One exception in this scenario is the resource allocation (minimum span algorithm), which must be executed in the data center hypervisor and directly benefits the cloud vendor with the increased utilization of available resources.

The second scenario is where cloud vendors voluntarily execute all algorithms in the hypervisor of each node. Conditions may be included to the multilateral federation agreements since the increased throughput and efficiently utilized resources would benefit all parties. Another motivation for the smaller vendors can be demand from the users who prefer large multi-location vendors such as Amazon, Google, or Microsoft because of the inter-cloud deployment ability they would offer.

6.4 Impact

Findings in this thesis are beneficial for the adoption and upcoming transformation of cloud computing. Better service quality and lower costs would attract more users to cloud services. Better utilization of cloud and network resources as well as increased throughput would also benefit infrastructure providers. In addition to the abovementioned novel aspects of the algorithms, the thesis study is also the first attempt to address resource mapping for both computation and storage elements of a virtual service as a single optimization problem.

This study is aimed to be a small step towards the vision of the cloud computing to make computation the fifth public utility (after water, electricity, gas, and telephony) which is easily accessible to everyone. This endeavour would also have a social impact towards the democratization of technology with more affordable, user-friendly software services as well as increased participation of users to the development of such services.

REFERENCES

[1] **Aral, A.**, (2015), RalloCloud, `https://github.com/atary/RalloCloud`.

[2] **Aral, A. and Ovatman, T.** (2016). A Decentralized Replica Placement Algorithm for Edge Computing, *IEEE Transactions on Parallel and Distributed Systems*, under review as of 19.09.2016.

[3] **Sosinsky, B.** (2010). *Cloud Computing Bible*, John Wiley & Sons.

[4] **Aral, A. and Ovatman, T.** (2016). Network-Aware Embedding of Virtual Machine Clusters onto Federated Cloud Infrastructure, *Journal of Systems and Software*, *120*, 89–104.

[5] **Campanella, M. and Farina, F.** (2014). The FEDERICA infrastructure and experience, *Computer Networks*, *61*, 176–183.

[6] **Armbrust, M.**, **Fox, A.**, **Griffith, R.**, **Joseph, A.D.**, **Katz, R.**, **Konwinski, A.**, **Lee, G.**, **Patterson, D.**, **Rabkin, A.**, **Stoica, I. and Zaharia, M.** (2010). A view of cloud computing, *Commun. ACM*, *53*(4), 50–58.

[7] **Mell, P. and Grance, T.** (2011). The NIST definition of cloud computing (draft), *NIST special publication*, *800*(145), 7.

[8] **Buyya, R.**, **Broberg, J. and Goscinski, A.M.** (2011). *Cloud computing: Principles and paradigms*, volume 87, Wiley.

[9] **Zhang, Q.**, **Cheng, L. and Boutaba, R.** (2010). Cloud Computing: State-of-the-art and Research Challenges, *Journal of Internet Services and Applications*, *1*(1), 7–18.

[10] **Papagianni, C.**, **Leivadeas, A.**, **Papavassiliou, S.**, **Maglaris, V.**, **Cervello-Pastor, C. and Monje, A.** (2013). On the Optimal Allocation of Virtual Resources in Cloud Computing Networks, *IEEE Transactions on Computers*, *62*(6), 1060–1071.

[11] **McKay, B.D.** (1981). *Practical graph isomorphism*, Department of Computer Science, Vanderbilt University Tennessee, US.

[12] **Garey, M.R. and Johnson, D.S.** (2002). *Computers and intractability*, volume 29, wh freeman New York.

[13] **Solnon, C.** (2010). Alldifferent-based filtering for subgraph isomorphism, *Artificial Intelligence*, *174*(12), 850–864.

[14] **Melo, M.T.**, **Nickel, S. and Saldanha-Da-Gama, F.** (2009). Facility Location and Supply Chain Management – A Review, *European Journal of Operational Research*, *196*(2), 401–412.

[15] **Owen, S.H. and Daskin, M.S.** (1998). Strategic Facility Location: A Review, *European Journal of Operational Research*, *111*(3), 423–447.

[16] **Guha, S. and Khuller, S.** (1999). Greedy Strikes Back: Improved Facility Location Algorithms, *Journal of Algorithms*, *31*(1), 228–248.

[17] **Buyya, R.**, **Yeo, C.S.**, **Venugopal, S.**, **Broberg, J. and Brandic, I.** (2009). Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, *Future Generation Computer Systems*, *25*(6), 599–616.

[18] **Armbrust, M.**, **Fox, A.**, **Griffith, R.**, **Joseph, A.D.**, **Katz, R.**, **Konwinski, A.**, **Lee, G.**, **Patterson, D.**, **Rabkin, A.**, **Stoica, I.** *et al.* (2010). A View of Cloud Computing, *Communications of the ACM*, *53*(4), 50–58.

[19] **Sotiriadis, S.**, **Bessis, N.**, **Kuonen, P. and Antonopoulos, N.** (2013). The inter-cloud meta-scheduling (ICMS) framework, *Proceedings of the IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, IEEE, pp.64–73.

[20] **Erdil, D.C.** (2013). Autonomic cloud resource sharing for intercloud federations, *Future Generation Computer Systems*, *29*(7), 1700–1708.

[21] **Lewis, G.A.** (2013). Role of Standards in Cloud-Computing Interoperability, *Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS)*, IEEE, pp.1652–1661.

[22] **Lucas-Simarro, J.L.**, **Moreno-Vozmediano, R.**, **Montero, R.S. and Llorente, I.M.** (2013). Scheduling strategies for optimal service deployment across multiple clouds, *Future Generation Computer Systems*, *29*(6), 1431–1441.

[23] **Ferry, N.**, **Rossini, A.**, **Chauvel, F.**, **Morin, B. and Solberg, A.** (2013). Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems, *Proceedings of the IEEE Sixth International Conference on Cloud Computing (CLOUD)*, IEEE, pp.887–894.

[24] **Barker, A.**, **Varghese, B. and Thai, L.** (2015). Cloud Services Brokerage: A Survey and Research Roadmap, *Proceedings of the IEEE 8th International Conference on Cloud Computing*, IEEE, pp.1029–1032.

[25] **Tordsson, J.**, **Montero, R.S.**, **Moreno-Vozmediano, R. and Llorente, I.M.** (2012). Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers, *Future Generation Computer Systems*, *28*(2), 358–367.

[26] **Cucinotta, T.**, **Lugones, D.**, **Cherubini, D. and Oberle, K.** (2014). Brokering SLAs for End-to-End QoS in Cloud Computing., *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER)*, pp.610–615.

[27] **Aceto, G.**, **Botta, A.**, **De Donato, W. and Pescapè, A.** (2013). Cloud monitoring: A survey, *Computer Networks*, *57*(9), 2093–2115.

[28] **Iosup, A.**, **Prodan, R. and Epema, D.**, (2014). Iaas cloud benchmarking: approaches, challenges, and experience, Cloud Computing for Data-Intensive Applications, Springer, pp.83–104.

[29] **Varghese, B.**, **Akgun, O.**, **Miguel, I.**, **Thai, L. and Barker, A.** (2014). Cloud benchmarking for performance, *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, IEEE, pp.535–540.

[30] **Alhazmi, K.**, **Abu Sharkh, M.**, **Ban, D. and Shami, A.** (2014). A map of the clouds: virtual network mapping in cloud computing data centers, *Proceedings of the 27th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, IEEE, pp.1–6.

[31] **Alicherry, M. and Lakshman, T.** (2012). Network aware resource allocation in distributed clouds, *Proceedings of the 31st Annual IEEE International Conference on Computer Communications (INFOCOM)*, IEEE, pp.963–971.

[32] **Altmann, J. and Kashef, M.M.** (2014). Cost model based service placement in federated hybrid clouds, *Future Generation Computer Systems*, *41*, 79–90.

[33] **De Falco, I.**, **Scafuri, U. and Tarantino, E.** (2014). Two new fast heuristics for mapping parallel applications on cloud computing, *Future Generation Computer Systems*, *37*, 1–13.

[34] **Konstanteli, K.**, **Cucinotta, T.**, **Psychas, K. and Varvarigou, T.A.** (2014). Elastic admission control for federated cloud services, *IEEE Transactions on Cloud Computing*, *2*(3), 348–361.

[35] **Leivadeas, A.**, **Papagianni, C. and Papavassiliou, S.** (2013). Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning, *IEEE Transactions on Parallel and Distributed Systems*, *24*(6), 1077–1086.

[36] **Papagianni, C.**, **Leivadeas, A.**, **Papavassiliou, S.**, **Maglaris, V.**, **Cervello-Pastor, C. and Monje, A.** (2013). On the optimal allocation of virtual resources in cloud computing networks, *IEEE Transactions on Computers*, *62*(6), 1060–1071.

[37] **Pittaras, C.**, **Papagianni, C.**, **Leivadeas, A.**, **Grosso, P.**, **van der Ham, J. and Papavassiliou, S.** (2015). Resource discovery and allocation for federated virtualized infrastructures, *Future Generation Computer Systems*, *42*, 55–63.

[38] **Xin, Y.**, **Baldine, I.**, **Mandal, A.**, **Heermann, C.**, **Chase, J. and Yumerefendi, A.** (2011). Embedding virtual topologies in networked clouds, *Proceedings of the 6th International Conference on Future Internet Technologies*, ACM, pp.26–29.

[39] **Chowdhury, M., Rahman, M.R. and Boutaba, R.** (2012). Vineyard: Virtual network embedding algorithms with coordinated node and link mapping, *IEEE/ACM Transactions on Networking (TON)*, *20*(1), 206–219.

[40] **Lischka, J. and Karl, H.** (2009). A virtual network mapping algorithm based on subgraph isomorphism detection, *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, ACM, pp.81–88.

[41] **Wei, X., Li, H., Yang, K. and Zou, L.** (2014). Topology-aware Partial Virtual Cluster Mapping Algorithm on Shared Distributed Infrastructures, *IEEE Transactions on Parallel and Distributed Systems*, *25*(10), 2721–2730.

[42] **Zhang, Z., Su, S., Lin, Y., Cheng, X., Shuang, K. and Xu, P.** (2015). Adaptive multi-objective artificial immune system based virtual network embedding, *Journal of Network and Computer Applications*, *53*, 140–155.

[43] **Houidi, I., Louati, W., Ameur, W.B. and Zeghlache, D.** (2011). Virtual network provisioning across multiple substrate networks, *Computer Networks*, *55*(4), 1011–1023.

[44] **Konstanteli, K., Cucinotta, T. and Varvarigou, T.** (2010). Optimum allocation of distributed service workflows with probabilistic real-time guarantees, *Service Oriented Computing and Applications*, *4*(4), 229–243.

[45] **Zaheer, F.E., Xiao, J. and Boutaba, R.** (2010). Multi-provider service negotiation and contracting in network virtualization, *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, IEEE, pp.471–478.

[46] **Biran, O., Corradi, A., Fanelli, M., Foschini, L., Nus, A., Raz, D. and Silvera, E.** (2012). A stable network-aware vm placement for cloud systems, *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE, pp.498–506.

[47] **Cheng, X., Su, S., Zhang, Z., Wang, H., Yang, F., Luo, Y. and Wang, J.** (2011). Virtual network embedding through topology-aware node ranking, *ACM SIGCOMM Computer Communication Review*, *41*(2), 38–47.

[48] **LaCurts, K., Deng, S., Goyal, A. and Balakrishnan, H.** (2013). Choreo: Network-aware task placement for cloud applications, *Proceedings of the Internet Measurement Conference (IMC)*, ACM, pp.191–204.

[49] **Li, X., Wang, H., Ding, B., Li, X. and Feng, D.** (2014). Resource allocation with multi-factor node ranking in data center networks, *Future Generation Computer Systems*, *32*, 1–12.

[50] **Sun, G., Yu, H., Anand, V. and Li, L.** (2013). A cost efficient framework and algorithm for embedding dynamic virtual network requests, *Future Generation Computer Systems*, *29*(5), 1265–1277.

[51] **Alicherry, M. and Lakshman, T.** (2013). Optimizing data access latencies in cloud systems by intelligent virtual machine placement, *Proceedings of the 32nd Annual IEEE International Conference on Computer Communications (INFOCOM)*, IEEE, pp.647–655.

[52] **Moschakis, I.A. and Karatza, H.D.** (2015). Multi-criteria scheduling of Bag-of-Tasks applications on heterogeneous interlinked clouds with simulated annealing, *Journal of Systems and Software*, *101*, 1–14.

[53] **Thai, L.**, **Barker, A.**, **Varghese, B.**, **Akgun, O. and Miguel, I.** (2014). Optimal deployment of geographically distributed workflow engines on the cloud, *Proceedings of the IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, pp.811–816.

[54] **Mell, P. and Grance, T.** (2011). The NIST definition of cloud computing, **Technical Report**, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg.

[55] **Endo, P.**, **de Almeida Palhares, A.**, **Pereira, N.**, **Goncalves, G.**, **Sadok, D.**, **Kelner, J.**, **Melander, B. and Mangs, J.E.** (2011). Resource allocation for distributed cloud: concepts and research challenges, *IEEE Network*, *25*(4), 42–46.

[56] **Xiao, Z.**, **Song, W. and Chen, Q.** (2013). Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment, *IEEE Transactions on Parallel and Distributed Systems*, *24*(6), 1107–1117.

[57] **Yang, K.**, **Gu, J.**, **Zhao, T. and Sun, G.** (2011). An Optimized Control Strategy for Load Balancing Based on Live Migration of Virtual Machine, *Sixth Annual Chinagrid Conference (ChinaGrid)*, pp.141–146.

[58] **Wang, Y.**, **Chen, S. and Pedram, M.** (2013). Service Level Agreement-Based Joint Application Environment Assignment and Resource Allocation in Cloud Computing Systems, *Green Technologies Conference, 2013 IEEE*, pp.167–174.

[59] **Karlsson, M. and Karamanolis, C.** (2004). Choosing replica placement heuristics for wide-area systems, *Proceedings of the 24th International Conference on Distributed Computing Systems*, IEEE, pp.350–359.

[60] **Ma, J.**, **Liu, W. and Glatard, T.** (2013). A classification of file placement and replication methods on grids, *Future Generation Computer Systems*, *29*(6), 1395–1406.

[61] **Amjad, T.**, **Sher, M. and Daud, A.** (2012). A survey of dynamic replication strategies for improving data availability in data grids, *Future Generation Computer Systems*, *28*(2), 337–349.

[62] **Grace, R.K. and Manimegalai, R.** (2014). Dynamic replica placement and selection strategies in data grids—a comprehensive survey, *Journal of Parallel and Distributed Computing*, *74*(2), 2099–2108.

[63] **Agarwal, S.**, **Dunagan, J.**, **Jain, N.**, **Saroiu, S.**, **Wolman, A. and Bhogan, H.** (2010). Volley: Automated Data Placement for Geo-Distributed Cloud Services., *NSDI*, volume 10, pp.28–0.

[64] **Andronikou, V.**, **Mamouras, K.**, **Tserpes, K.**, **Kyriazis, D. and Varvarigou, T.** (2012). Dynamic QoS-aware data replication in grid environments based on data "importance", *Future Generation Computer Systems*, *28*(3), 544–553.

[65] **Ali, M.**, **Bilal, K.**, **Khan, S.**, **Veeravalli, B.**, **Li, K. and Zomaya, A.** (2015). DROPS: Division and Replication of Data in the Cloud for Optimal Performance and Security, *IEEE Transactions on Cloud Computing*.

[66] **Chen, F.**, **Guo, K.**, **Lin, J. and La Porta, T.** (2012). Intra-cloud lightning: Building CDNs in the cloud, *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, IEEE, pp.433–441.

[67] **Lizhen, L.C.**, **Zhang, J.**, **Yue, L.**, **Shi, Y.**, **Li, H. and Yuan, D.** (2015). A Genetic Algorithm Based Data Replica Placement Strategy for Scientific Applications in Clouds, *IEEE Transactions on Services Computing*.

[68] **Kalpakis, K.**, **Dasgupta, K. and Wolfson, O.** (2001). Optimal placement of replicas in trees with read, write, and storage costs, *IEEE Transactions on Parallel and Distributed Systems*, *12*(6), 628–637.

[69] **Lin, J.W.**, **Chen, C.H. and Chang, J.M.** (2013). QoS-aware data replication for data-intensive applications in cloud computing systems, *IEEE Transactions on Cloud Computing*, *1*(1), 101–115.

[70] **Zhang, Q.**, **Li, S.**, **Li, Z.**, **Xing, Y.**, **Yang, Z. and Dai, Y.** (2015). CHARM: A Cost-Efficient Multi-Cloud Data Hosting Scheme with High Availability, *IEEE Transactions on Cloud Computing*, *3*(3), 372–386.

[71] **Wu, S.**, **Li, K.C.**, **Mao, B. and Liao, M.** (2016). DAC: Improving storage availability with Deduplication-Assisted Cloud-of-Clouds, *Future Generation Computer Systems*.

[72] **Qiu, L.**, **Padmanabhan, V.N. and Voelker, G.M.** (2001). On the placement of web server replicas, *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, IEEE, pp.1587–1596.

[73] **Szymaniak, M.**, **Pierre, G. and Van Steen, M.** (2005). Latency-driven replica placement, *The 2005 Symposium on Applications and the Internet*, IEEE, pp.399–405.

[74] **Liu, P. and Wu, J.J.** (2006). Optimal replica placement strategy for hierarchical data grid systems, *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, volume 1, IEEE, pp.417–420.

[75] **Benoit, A.**, **Rehn-Sonigo, V. and Robert, Y.** (2008). Replica placement and access policies in tree networks, *IEEE Transactions on Parallel and Distributed Systems*, *19*(12), 1614–1627.

[76] **Rochman, Y.**, **Levy, H. and Brosh, E.** (2013). Resource placement and assignment in distributed network topologies, *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, IEEE, pp.1914–1922.

[77] **Deng, K.**, **Ren, K.**, **Zhu, M. and Song, J.** (2015). A Data and Task Co-scheduling Algorithm for Scientific Cloud Workflows, *IEEE Transactions on Cloud Computing*.

[78] **Douceur, J.R. and Wattenhofer, R.P.** (2001). Competitive hill-climbing strategies for replica placement in a distributed file system, *International Symposium on Distributed Computing*, Springer, pp.48–62.

[79] **Tang, M.**, **Lee, B.S.**, **Yeo, C.K. and Tang, X.** (2005). Dynamic replication algorithms for the multi-tier data grid, *Future Generation Computer Systems*, *21*(5), 775–790.

[80] **Tang, X. and Xu, J.** (2005). QoS-aware replica placement for content distribution, *IEEE Transactions on Parallel and Distributed Systems*, *16*(10), 921–932.

[81] **Khanli, L.M.**, **Isazadeh, A. and Shishavan, T.N.** (2011). PHFS: A dynamic replication method, to decrease access latency in the multi-tier data grid, *Future Generation Computer Systems*, *27*(3), 233–244.

[82] **Lee, M.C.**, **Leu, F.Y. and Chen, Y.p.** (2012). PFRF: An adaptive data replication algorithm based on star-topology data grids, *Future Generation Computer Systems*, *28*(7), 1045–1057.

[83] **Saadat, N. and Rahmani, A.M.** (2012). PDDRA: A new pre-fetching based dynamic data replication algorithm in data grids, *Future Generation Computer Systems*, *28*(4), 666–681.

[84] **Fan, X.**, **Ma, X.**, **Liu, J. and Li, D.** (2014). Dependency-aware data locality for MapReduce, *IEEE 7th International Conference on Cloud Computing*, IEEE, pp.408–415.

[85] **Hu, M.**, **Luo, J.**, **Wang, Y. and Veeravalli, B.** (2014). Practical resource provisioning and caching with dynamic resilience for cloud-based content distribution networks, *IEEE Transactions on Parallel and Distributed Systems*, *25*(8), 2169–2179.

[86] **Chen, C.A.**, **Won, M.**, **Stoleru, R. and Xie, G.G.** (2015). Energy-efficient fault-tolerant data storage and processing in mobile cloud, *IEEE Transactions on cloud computing*, *3*(1), 28–41.

[87] **Pantazopoulos, P.**, **Karaliopoulos, M. and Stavrakakis, I.** (2014). Distributed placement of autonomic internet services, *IEEE Transactions on Parallel and Distributed Systems*, *25*(7), 1702–1712.

[88] **Chen, Y.**, **Katz, R.H. and Kubiatowicz, J.D.** (2002). Dynamic replica placement for scalable content delivery, *International Workshop on Peer-to-Peer Systems*, Springer, pp.306–318.

[89] **Shen, H.** (2010). An efficient and adaptive decentralized file replication algorithm in P2P file sharing systems, *IEEE Transactions on Parallel and Distributed Systems*, *21*(6), 827–840.

[90] **Papagianni, C.**, **Leivadeas, A. and Papavassiliou, S.** (2013). A cloud-oriented content delivery network paradigm: Modeling and assessment, *IEEE Transactions on Dependable and Secure Computing*, *10*(5), 287–300.

[91] **Bonvin, N.**, **Papaioannou, T.G. and Aberer, K.** (2010). A self-organized, fault-tolerant and scalable replication scheme for cloud storage, *Proceedings of the 1st ACM symposium on Cloud computing*, ACM, pp.205–216.

[92] **Sashi, K. and Thanamani, A.S.** (2011). Dynamic replication in a data grid using a Modified BHR Region Based Algorithm, *Future Generation Computer Systems*, *27*(2), 202–210.

[93] **Liao, X.**, **Jin, H. and Yu, L.** (2012). A novel data replication mechanism in P2P VoD system, *Future Generation Computer Systems*, *28*(6), 930–939.

[94] **Liu, G.**, **Shen, H. and Chandler, H.** (2013). Selective data replication for online social networks with distributed datacenters, *21st IEEE International Conference on Network Protocols (ICNP)*, IEEE, pp.1–10.

[95] **Zaman, S. and Grosu, D.** (2011). A distributed algorithm for the replica placement problem, *IEEE Transactions on Parallel and Distributed Systems*, *22*(9), 1455–1468.

[96] **Smaragdakis, G.**, **Laoutaris, N.**, **Oikonomou, K.**, **Stavrakakis, I. and Bestavros, A.** (2014). Distributed server migration for scalable Internet service deployment, *IEEE/ACM Transactions on Networking*, *22*(3), 917–930.

[97] **Moscibroda, T. and Wattenhofer, R.** (2005). Facility location: distributed approximation, *Proceedings of the 24th ACM symposium on Principles of distributed computing*, ACM, pp.108–117.

[98] **Pandit, S. and Pemmaraju, S.** (2009). Return of the primal-dual: distributed metric facilitylocation, *Proceedings of the 28th ACM symposium on Principles of distributed computing*, ACM, pp.180–189.

[99] **Calheiros, R.N.**, **Ranjan, R.**, **Beloglazov, A.**, **De Rose, C.A. and Buyya, R.** (2011). CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, *Software: Practice and Experience*, *41*(1), 23–50.

[100] **Greenberg, A.**, **Hamilton, J.**, **Maltz, D.A. and Patel, P.** (2008). The cost of a cloud: research problems in data center networks, *ACM SIGCOMM computer communication review*, *39*(1), 68–73.

[101] **Rochwerger, B.**, **Breitgand, D.**, **Levy, E.**, **Galis, A.**, **Nagin, K.**, **Llorente, I.M.**, **Montero, R.**, **Wolfsthal, Y.**, **Elmroth, E.**, **Caceres, J.** *et al.* (2009). The reservoir model and architecture for open federated cloud computing, *IBM Journal of Research and Development*, *53*(4), 4–1.

[102] **Buyya, R.**, **Ranjan, R. and Calheiros, R.N.**, (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services, Algorithms and architectures for parallel processing, Springer, pp.13–31.

[103] **Grozev, N. and Buyya, R.** (2014). Inter-Cloud architectures and application brokering: taxonomy and survey, *Software: Practice and Experience*, *44*(3), 369–390.

[104] **Fischer, A.**, **Botero, J.F.**, **Till Beck, M.**, **De Meer, H. and Hesselbach, X.** (2013). Virtual network embedding: A survey, *IEEE Communications Surveys & Tutorials*, *15*(4), 1888–1906.

[105] **Dean, J. and Ghemawat, S.** (2008). MapReduce: simplified data processing on large clusters, *Communications of the ACM*, *51*(1), 107–113.

[106] **García-Valls, M.**, **Cucinotta, T. and Lu, C.** (2014). Challenges in real-time virtualization and predictable cloud computing, *Journal of Systems Architecture*, *60*(9), 726–740.

[107] **Amazon Web Services**, `https://aws.amazon.com/`, retrieval date: 15.02.2016.

[108] **Watts, D.J. and Strogatz, S.H.** (1998). Collective dynamics of small-world networks, *Nature*, *393*(6684), 440–442.

[109] **Bonomi, F.**, **Milito, R.**, **Zhu, J. and Addepalli, S.** (2012). Fog computing and its role in the internet of things, *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, ACM, pp.13–16.

[110] **Valancius, V.**, **Laoutaris, N.**, **Massoulié, L.**, **Diot, C. and Rodriguez, P.** (2009). Greening the internet with nano data centers, *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ACM, pp.37–48.

[111] **Satyanarayanan, M.**, **Bahl, P.**, **Caceres, R. and Davies, N.** (2009). The case for vm-based cloudlets in mobile computing, *IEEE pervasive Computing*, *8*(4), 14–23.

[112] **Banditwattanawong, T.**, **Masdisornchote, M. and Uthayopas, P.** (2016). Multi-provider cloud computing network infrastructure optimization, *Future Generation Computer Systems*, *55*, 116–128.

[113] **Wu, Z.**, **Butkiewicz, M.**, **Perkins, D.**, **Katz-Bassett, E. and Madhyastha, H.V.** (2013). Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services, *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ACM, pp.292–308.

[114] **Ranganathan, K. and Foster, I.** (2001). Identifying dynamic replication strategies for a high-performance data grid, *Grid Computing—GRID 2001*, 75–86.

[115] **Jin, S. and Bestavros, A.** (2001). GreedyDual* Web caching algorithm: exploiting the two sources of temporal locality in Web request streams, *Computer Communications*, *24*(2), 174–183.

[116] **Liao, J.**, **Trahay, F.**, **Xiao, G.**, **Li, L. and Ishikawa, Y.** (2015). Performing Initiative Data Prefetching in Distributed File Systems for Cloud Computing, *IEEE Transactions on Cloud Computing*.

[117] **The CAIDA UCSD Anonymized Internet Traces 2015 - [2015-02-19]**, `http://www.caida.org/data/passive/passive_2015_dataset.xml`, retrieval date: 07.01.2016.

[118] **Tanenbaum, A.S. and Van Steen, M.** (2007). *Distributed systems*, Prentice-Hall.

[119] **Medina, A.**, **Lakhina, A.**, **Matta, I. and Byers, J.** (2001). BRITE: An approach to universal topology generation, *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE, pp.346–353.

[120] **Barabási, A.L. and Albert, R.** (1999). Emergence of scaling in random networks, *Science*, *286*(5439), 509–512.

CURRICULUM VITAE

Name Surname: Atakan Aral

Place and Date of Birth: Istanbul, 1986

E-Mail: aralat@itu.edu.tr

EDUCATION:

- **B.Sc.:** 2009, Istanbul Technical University, Computer Engineering
- **M.Sc.:** 2011, Politecnico di Milano, Computer Science and Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2016–. . . Postdoctoral Researcher at Vienna University of Technology, Institute of Informatics
- 2011–2016 Research Assistant at Istanbul Technical University, Faculty of Computer and Informatics Engineering
- 2012–2016 The Scientific and Technological Research Council of Turkey (TUBITAK) Graduate Scholarship
- 2016 IEEE International Conference on Cloud Engineering (IC2E) Travel Grant
- 2014 NETAS Ph.D. Project Incentive Award
- 2012–2013 Informatics Association of Turkey Leaders of Technology Graduate Scholarship
- 2005–2009 Istanbul Technical University ARI Highest Success Awards
- 2005–2009 Prime Ministry of Turkey Premiership Scholarship

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Aral, A.**, Ovatman, T., 2017?. A Decentralized Replica Placement Algorithm for Edge Computing, *IEEE Transactions on Parallel and Distributed Systems,* (Under revision).
- **Aral, A.**, Ovatman, T., 2016. Network-Aware Embedding of Virtual Machine Clusters onto Federated Cloud Infrastructure, *The Journal of Systems and Software,* 120, 89–104.

- **Aral, A.**, 2016. Network-Aware Resource Allocation in Distributed Clouds, *In Doctoral Symposium of the IEEE International Conference on Cloud Engineering, IC2E 2016.*

- **Aral, A.**, Ovatman, T., 2015. Subgraph Matching for Resource Allocation in the Federated Cloud Environment, *In Proceedings of the IEEE International Conference on Cloud Computing, IEEE CLOUD 2015.*

- **Aral, A.**, Ovatman, T., 2014. Improving Resource Utilization in Cloud Environments Using Application Placement Heuristics, *In Proceedings of 4th International Conference on Cloud Computing and Services Science, CLOSER 2014.*

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- Ovatman, T., **Aral, A.**, Polat, D., Unver, A.O., 2014. An Overview of Model Checking Practices on Verification of PLC Software, *Journal of Software and Systems Modeling,* 15 (4), 937-–960

- **Aral, A.**, Ovatman, T., 2013. Utilization of Method Graphs to Measure Cohesion in Object Oriented Software, *In Proceedings of the 7th IEEE International Workshop on Quality Oriented Reuse of Software, QUORS 2013.*

- **Aral, A.**, Akin, I.Z., Brambilla, M., 2012. Mobile Multi-domain Search over Structured Web Data, *Search Computing - Broadening Web Search, Springer LNCS 7538.*